МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Львівський національний університет імені Івана Франка

Факультет електроніки та комп'ютерних технологій

ЗВІТ

З ЛАБОРАТОРНОЇ РОБОТИ № 1

з дисципліни **„Паттерни проектування”**

Виконав:

студент 2 курсу

групи ФЕП-23

Стецик Олег Степанович

Перевірив:

Доцент

Сінькевич Олег Олександрович

Львів-2024

Мета роботи: Реалізувати асоціації та агрегації між заданими класами. Необхідні базові знання з об'єктно-орієнтованого проектування. Ви можете використовувати довільні мови програмування, які підтримують ООП. Не забувайте правильно коментувати свій код.

Зміст роботи:

Main.py

```python
from customers import Customer
from operators import Operator
from bills import Bill


def main():
    # ініціалізація операторів
    operators = [Operator(0, 0.5, 0.1, 0.2, 10), Operator(1, 0.6, 0.2, 0.25, 5)]

    # ініціалізація рахунків
    bills = [Bill(1000), Bill(500)]

    customers = [Customer(0, 'Олег', 'Стецик', 18, operators, bills),
                 Customer(1, 'Олег', 'Школик', 19, operators, bills)]

    #виклик дій для клієнтів
    customers[0].talk(10, customers[1], 0)  #Рома говорить з Антон
    customers[1].message(5, customers[0], 1)  # Антон відправляє повідомлення Ромі
    customers[0].connection(100, 0)    #Рома використовує інтернет

   # оплата рахунку
    customers[0].get_bill(0).pay(200)

    # зміна ліміту рахунку
    customers[0].get_bill(0).change_limit(200)


if __name__ == "__main__":
    main()   #щоб запускався як головна програма
```

customers.py

```python
from multiprocessing.managers import Value
from typing import List, Self
from bills import Bill
from operators import Operator


class Customer:
    #клас customer представляє клієнта системи зв'язку
```

```
    def __init__(self, id: int, first_name: str, last_name: str,
                 age: int, operators: List[Operator], bills: List[Bill],
limiting_amount: float = 1000.0) -> None:
        # конструктор
        self.id: int = id
        self.first_name: str = first_name
        self.last_name: str = last_name
        self._age: int = age
        self.operators: List[Operator] = operators
        self.bills: List[Bill] = bills
        self.limiting_amount: float = limiting_amount

    def talk(self, minutes: float, customer: Self, operator_id: int) -> None:
        #метод для звінку
        operator = self.operators[operator_id]
        talk_cost = operator.calc_talking_cost(minutes, self)

        bill = self.bills[operator_id]
        if not bill.check():
            bill.add_debt(talk_cost)
            print(f"{self.first_name} говорив з {customer.first_name}  {minutes}
хвилин.")
        else:
            print(f"{self.first_name} перевищив ліміт рахунку і не може виконати
дзвінок.")

    def message(self, quantity: int, customer: Self, operator_id: int) -> None:
        #метод для відправлення повідомлення
        operator = self.operators[operator_id]
        message_cost = operator.calc_message_cost(quantity, self, customer)

        bill = self.bills[operator_id]
        if not bill.check():
            bill.add_debt(message_cost)
            print(f"{self.first_name} відправив {quantity} повідомлень для
{customer.first_name}.")
        else:
            print(f"{self.first_name} перевищив ліміт рахунку і не може надіслати
повідомлення.")

    def connection(self, amount: float, operator_id: int) -> None:
       # метод підключення до інтернету
        operator = self.operators[operator_id]
        network_cost = operator.calc_network_cost(amount)

        bill = self.bills[operator_id]
        if not bill.check():
            bill.add_debt(network_cost)
            print(f"{self.first_name} використав {amount} МВ.")
        else:
```

```
            print(f"{self.first_name} перевищив ліміт рахунку і не може
використовувати інтернет.")



    @property
    def age(self) -> int:
        return self._age

    @age.setter
    def age(self, age: int) -> None:
        if age <= 0:
            raise ValueError(f'Age must be positive')
        self._age = age

    def get_operator(self, operator_id: int) -> Operator:
        return self.operators[operator_id]

    def set_operator(self, operator_id: int, operator: Operator) -> None:
        self.operators[operator_id] = operator

    def get_bill(self, operator_id: int) -> Bill:
        return self.bills[operator_id]

    def set_bill(self, operator_id: int, bill: Bill) -> None:
        self.bills[operator_id] = bill
```

Bills.py

```
class Bill:
    #клас bill представляє рахунок клієнта за послуги оператора

    def __init__(self, limiting_amount: float) -> None:
        #конструктор для класу Bill
        self.limiting_amount: float = limiting_amount            #макс сума боргу
        self.current_debt: float = 0.0

    def check(self) -> bool:
        #перевірка чи перевищено ліміт
        return self.current_debt >= self.limiting_amount
```

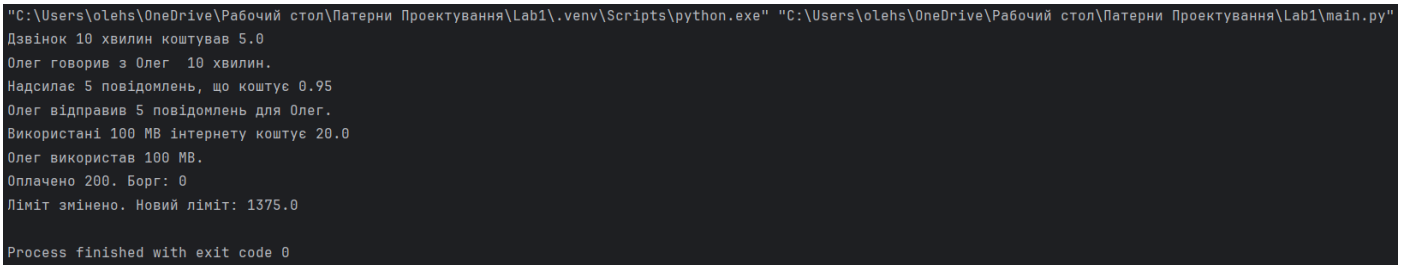
```
    def add_debt(self, debt: float) -> None:
        #додавання боргу до рахунку
        tentative_debt = debt + self.current_debt                #попередній -
tentative
        if tentative_debt <= self.limiting_amount:
            self.current_debt += debt
        else:
            print(f"Ви перевищили ліміт! Ваш борг становитиме {tentative_debt}")

    def pay(self, amount: float) -> None:
        #оплата рахунку
        self.current_debt -= amount
        if self.current_debt < 0:
            self.limiting_amount += abs(self.current_debt)
            self.current_debt = 0
        print(f"Оплачено {amount}. Борг: {self.current_debt}")

    def change_limit(self, amount: float) -> None:
        #зміна ліміту рахунку
        self.limiting_amount += amount
        print(f"Ліміт змінено. Новий ліміт: {self.limiting_amount}")

    def get_limiting_amount(self) -> float:
        return self.limiting_amount

    def get_current_debt(self) -> float:
        return self.current_debt
```

Operators.py

```
from typing import TYPE_CHECKING

if TYPE_CHECKING:
 from customers import Customer


class Operator:
 #клас operator представляє оператора зв'язку та тарифи на послуги

def __init__(self, id: int, talking_charge: float, message_cost: float,
 network_charge: float, discount_rate: int) -> None:
 #конструктор
 self.id = id
```

```
 self.talking_charge: float = talking_charge
 self.message_cost: float = message_cost
 self.network_charge: float = network_charge
 self.discount_rate: int = discount_rate

def calc_talking_cost(self, minutes: float, customer: 'Customer') -> float:
 #метод розрахунку вартості дзвінка
 cost = self.talking_charge * minutes
 if customer.age < 18 or customer.age > 65:
 discount = cost * (self.discount_rate / 100)
 cost -= discount
 print(f"Дзвінок {minutes} хвилин коштував {cost}")
 return cost

def calc_message_cost(self, quantity: int, customer: 'Customer', other:
'Customer') -> float:
 #метод розрахунку вартості повідомлень
 cost = self.message_cost * quantity
 if self.id == other.operators[self.id].id: # той самий оператор
 discount = cost * (self.discount_rate / 100)
 cost -= discount
 print(f"Надсилає {quantity} повідомлень, що коштує {cost}")
 return cost

def calc_network_cost(self, amount: float) -> float:
 #метод для розрахунку вартості інтернету
 cost = self.network_charge * amount
 print(f"Використані {amount} МВ інтернету коштує {cost}")
 return cost


def get_talking_charge(self) -> float:
 return self.talking_charge

def set_talking_charge(self, charge: float) -> None:
 self.talking_charge = charge

def get_message_cost(self) -> float:
 return self.message_cost

def set_message_cost(self, cost: float) -> None:
 self.message_cost = cost

def get_network_charge(self) -> float:
 return self.network_charge

def set_network_charge(self, charge: float) -> None:
 self.network_charge = charge

def get_discount_rate(self) -> int:
 return self.discount_rate

def set_discount_rate(self, rate: int) -> None:
 self.discount_rate = rate
```

Вивід:



Висновок: Під час виконання лабораторної роботи було створено систему для моделювання взаємодії клієнтів з операторами зв'язку. Реалізовано класи Customer, Operator та Bill, які відповідають за обробку дзвінків, повідомлень та інтернет-послуг.