Технології програмування

Лабораторна робота №3

**ООП**

**Мета роботи**: Використовуючи теоретичне підґрунтя про ООП у мові Python переробити програму телефонного довідника студентів використовуючи принципи ООП для формування відомостей про студентів.

**Хід роботи**

У ході виконання Лабораторної роботи №3 було здійснено комплексну переробку функціоналу телефонного довідника студентів групи, розробленого у Лабораторній роботі №2. Основні кроки переробки включали в себе впровадження принципів об'єктно-орієнтованого програмування (ООП).

1.Створено клас StudentList, який замість словників тепер містить об'єкти класу Student. Реалізовані методи для додавання нового запису, видалення існуючого та зміни даних. Впроваджено сортування списку студентів за ім'ям при додаванні нового запису.

2.Створено клас Utils для роботи з файлами, який має методи для зчитування та збереження інформації з файлів.

3.Замість словників, використання об'єктів класу Student забезпечує більш структурований та зрозумілий код.

4. Було описано всі класив окремих файлах з відповідними іменами, що полегшує розуміння коду та його підтримку.

5.Основний функціонал програми покритий юніт тестами, що дозволяє перевірити коректність роботи кожного фрагмента коду та забезпечити його стабільність.

**Текст програми**

**StudentList.py**

from Student import Student

class StudentList:

    def \_\_init\_\_(self):

        self.students = []

    def \_\_str\_\_(self):

        return '\n'.join(str(student) for student in self.students)

    def add\_new\_element(self):

        name = input("Please enter student name: ")

        phone = input("Please enter student phone: ")

        age = input("Please enter student age: ")

        grade = input("Please enter student grade: ")

        new\_student = Student(name, phone, age, grade)

        insert\_position = 0

        for student in self.students:

            if name > student.name:

                insert\_position += 1

            else:

                break

        self.students.insert(insert\_position, new\_student)

        print("New element has been added")

    def delete\_element(self):

        name = input("Please enter name to be deleted: ")

        for student in self.students:

            if name == student.name:

                self.students.remove(student)

                print(f"{name} has been deleted")

                return

        print("Element was not found")

    def update\_element(self):

        name = input("Please enter name to be updated: ")

        for student in self.students:

            if name == student.name:

                self.students.remove(student)

                new\_name = input("Enter new name: ")

                new\_phone = input("Enter new phone: ")

                new\_age = input("Enter new age: ")

                new\_grade = input("Enter new grade: ")

                updated\_student = Student(new\_name, new\_phone, new\_age, new\_grade)

                insert\_position = 0

                for s in self.students:

                    if new\_name > s.name:

                        insert\_position += 1

                    else:

                        break

                self.students.insert(insert\_position, updated\_student)

                print("Element has been updated")

                return

        print("Element was not found")

**Student.py**

class Student:

    def \_\_init\_\_(self, name, phone, age, grade):

        self.name = name

        self.surname = phone

        self.age = age

        self.phone = grade

    def \_\_str\_\_(self):

        return f"Student name: {self.name}, Student phone: {self.phone}, Student age: {self.age}, Grade: {self.grade}"

**lab03.py**

from Utils import Utils

import sys

def main():

    if len(sys.argv) != 2:

        print("Python script.py <file\_name.csv>")

        sys.exit(1)

    file\_name = sys.argv[1]

    student\_list = Utils.load\_data(file\_name)

    while True:

        choice = input("Please specify the action [ C create, U update, D delete, P print,  X exit ] ")

        match choice:

            case "C" | "c":

                print("New element will be created:")

                list.addNewElement(student\_list)

                list.printAllList(student\_list)

            case "U" | "u":

                print("Existing element will be updated:")

                list.updateElement(student\_list)

                list.printAllList(student\_list)

            case "D" | "d":

                print("Element will be deleted")

                list.deleteElement(student\_list)

            case "P" | "p":

                print("List will be printed")

                list.printAllList(student\_list)

            case "X" | "x":

                print("Exit")

                list.save\_data(file\_name, student\_list)

                break

            case \_:

                print("Wrong choice")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**lab3\_sample\_test.py**

import unittest

from Student import Student

from StudentList import StudentList

from Utils import Utils

from unittest import mock

class Testlab03(unittest.TestCase):

    def test\_load\_data(self):

        test\_file = "test03.csv"

        with open(test\_file, "w") as file:

            file.write("name,phone,age,grade\nLiza,12323232,33,98")

        student\_list = Utils.load\_data(test\_file)

        self.assertEqual(len(student\_list.students), 1)

    @mock.patch("builtins.input", side\_effect=["Liza", "12323232", "33", "98"])

    def test\_save\_data(self, mock\_input):

        test\_file = "test\_file.csv"

        student\_list = StudentList()

        student\_list.add\_student(Student("Liza", "12323232", "33", "98"))

        Utils.save\_data(test\_file, student\_list)

        with open(test\_file, "r") as file:

            content = file.read()

            self.assertIn("Liza", content)

class TestStudentListMethods(unittest.TestCase):

    @mock.patch("builtins.input", side\_effect=["Liza", "12323232", "33", "98"])

    def test\_addNewElement(self, mock\_input):

        student\_list = StudentList()

        student\_list.addNewElement()

        self.assertEqual(len(student\_list.students), 1)

        self.assertEqual(student\_list.students[0].name, "Liza")

    @mock.patch("builtins.input", return\_value="Liza")

    def test\_deleteElement(self, mock\_input):

        student\_list = StudentList()

        student\_list.addNewElement()

        student\_list.deleteElement()

        self.assertEqual(len(student\_list.students), 0)

    @mock.patch("builtins.input", side\_effect=["Liza", "Bob", "662622", "22", "88"])

    def test\_updateElement(self, mock\_input):

        student\_list = StudentList()

        student\_list.addNewElement()

        student\_list.updateElement()

        self.assertEqual(student\_list.students[0].name, "Bob")

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()