Міністерство освіти і науки України

Національний університет «Львівська політехніка»



**ЗВІТ**

з лабораторної роботи №9

з дисципліни: “Кросплатформні засоби програмування”

на тему: “Основи об’єктно-орієнтованого програмування у Python”

Виконав:

ст. гр. КІ-305

Ніколенко О.В.

Прийняв:

Іванов Ю.С.

Львів – 2023

**Мета:** оволодіти навиками реалізації парадигм об’єктно-орієнтованого програмування використовуючи засоби мови Python.

**Завдання:**

1. Написати та налагодити програму на мові Python згідно варіанту. Програма має задовольняти наступним вимогам:
   * класи програми мають розміщуватися в окремих модулях в одному пакеті;
   * точка входу в програму (main) має бути в окремому модулі;
   * мають бути реалізовані базовий і похідний класи предметної області згідно варіанту;
   * програма має містити коментарі.
2. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.
3. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації та завантажити його у ВНС.
4. Дати відповідь на контрольні запитання.

**Варіант № 20**


**Лістинг програми:**

main.py

from Boots import Boots

def main():

try:

martens = Boots("Dr.Martens 1460", 39, "leather")

martens.start()

martens.show\_info()

martens.show\_clean()

martens.insulation\_on\_boots()

martens.change\_lacing(2)

martens.show\_lacing()

martens.take\_off\_insulation()

martens.show\_size()

martens.end()

martens.start()

martens.show\_clean()

martens.show\_repair()

martens.repair()

martens.show\_repair()

martens.show\_material()

martens.clean()

martens.end()

martens.show\_clean()

martens.dispose()

except Exception as e:

print(e)

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

main()

Boots.py

from Shoes import Shoes

class IBoots:

def take\_off\_insulation(self):

pass

def insulation\_on\_boots(self):

pass

class Boots(Shoes, IBoots):

def \_\_init\_\_(self, name, size, material):

super().\_\_init\_\_(name, size, material)

self.insulation = False

def take\_off\_insulation(self):

if self.insulation:

self.insulation = False

print("Taking off insulation")

else:

print("Insulation is already removed")

def insulation\_on\_boots(self):

if not self.insulation:

self.insulation = True

print("The insulation is on")

Shoes.py

class Shoes:

def \_\_init\_\_(self, name, size, material):

self.is\_put\_on = False

self.size = size

self.material = material

self.lacing\_types = ["Hash", "Twistie", "Riding bow", "Lattice", "Zipper"]

self.lacing = "Hash"

self.washed = Wash()

self.repaired = Repair()

self.name = name

self.count = 0

if size > 37:

self.count += 1

def start(self):

if not self.is\_put\_on:

self.is\_put\_on = True

self.washed.weared()

self.repaired.weared()

print("Put on")

else:

print("You already put on these shoes")

def end(self):

if self.is\_put\_on:

self.is\_put\_on = False

print("Take off")

else:

print("You took off those shoes")

def show\_lacing(self):

print(f"{self.lacing} lacing")

def show\_material(self):

print(f"Your shoes are made of {self.material}")

def change\_lacing(self, lacing\_type):

self.lacing = self.lacing\_types[lacing\_type - 1]

print(f"The lacing type changed to: {self.lacing}")

def show\_size(self):

print(f"Shoe size is: {self.size}")

def show\_info(self):

print(f"Name: {self.name} ; Size: {self.size} ; Material: {self.material}")

def show\_clean(self):

print(self.washed.is\_washed())

def clean(self):

return self.washed.clean()

def show\_repair(self):

print(self.repaired.is\_repaired())

def repair(self):

return self.repaired.repair1()

def dispose(self):

pass # Release any resources

class Wash:

def \_\_init\_\_(self):

self.washed = 100

def is\_washed(self):

return f"Your shoes are {self.washed}% clean"

def clean(self):

self.washed = 100

return "Your shoes are now clean"

def weared(self):

self.washed -= 2

if self.washed <= 0:

self.washed = 0

class Repair:

def \_\_init\_\_(self):

self.repaired = 100

def is\_repaired(self):

return f"Your shoes are {self.repaired}% unharmed"

def repair1(self):

self.repaired = 100

return "Your shoes have been repaired"

def weared(self):

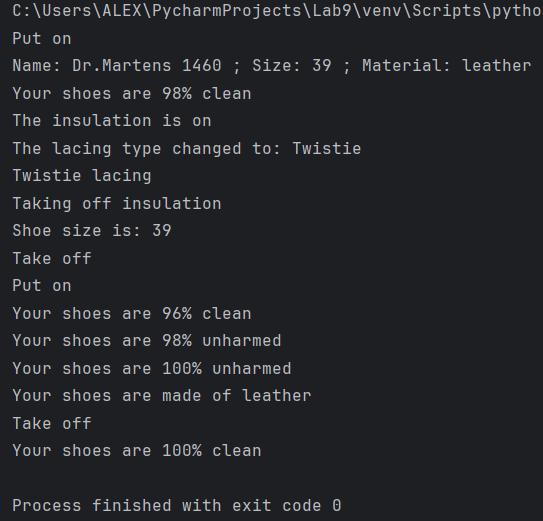
self.repaired -= 1

if self.repaired <= 0:

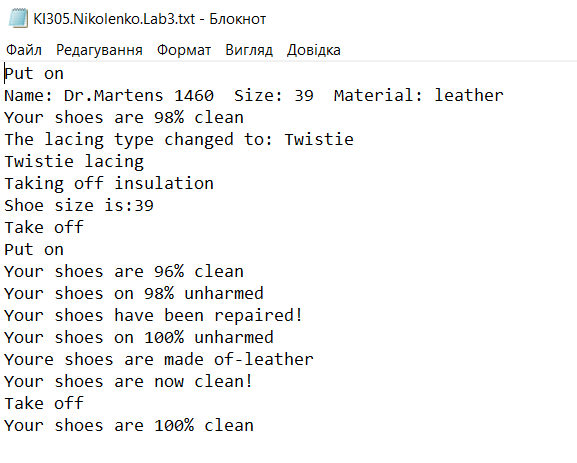
self.repaired = 0

print("Your shoes are beyond repair :(")

**Результат виконання :**



*Рис.1.Результат виводу на консоль*

**

*Рис.2.Результат виводу у файл*

**Відповіді на контрольні запитання:**

1. **Що таке модулі?**

Модулем у Python називається файл з розширенням \*.py. Ці файли можуть містити звичайні скрипти, змінні, функції, класи і їх комбінації.

1. **Як імпортувати модуль?**

Для цього використовуються два оператори – import та from-import.

1. **Як оголосити клас?**

Клас оголошується за допомогою ключового слова class після якого йде назва класу.

1. **Що може міститися у класі?**

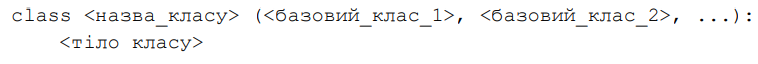
* дані, які належать класу (статичні дані-члени класу);
* дані, які належать об’єкту класу;
* методи, які належать класу (статична методи);
* методи, які належать об’єкту класу.

1. **Як називається конструктор класу?**

Роль конструктора відіграє метод \_\_init\_\_(self, ).

1. **Як здійснити спадкування?**

Для реалізації спадкування класи, які слід успадкувати вказуються у круглих дужках через кому після назви класу, який оголошується:

****

1. **Які види спадкування існують?**

* Одинарне спадкування
* Багатократне спадкування
* Інтерфейси
* Абстрактні класи
* Множинне спадкування
* Заміщення

1. **Які небезпеки є при множинному спадкуванні, як їх уникнути?**

* Амбігвітність – уникнути можна забезпечивши чітку структуру спадкування
* Складність ієрархій класів – уникнути можна дотримуючись принципу “розділення відповідальностей”
* Залежність від порядку – уникнути можна спроектувавши класи так, вони не залежали від порядку успадкування
* Дублювання коду – уникнути можна використовуючи абстрактні класи або інтерфейси для виділення спільного функціоналу та розділення його між класами
* Складність відлагодження та обслуговування – уникнути можна називаючи класи та методи відповідно до їх функції

1. **Що таке класи-домішки?**

Домішки або Mixin – це шаблон проектування, в якому деякий метод базового класу використовує метод, який не визначається у цьому класі. Цей метод призначений для реалізації іншим базовим класом. Клас-домішка або mixin class – це клас, який використовується у цьому шаблоні, надаючи функціональні можливості (методи), але не призначений для самостійного використання у вигляді об’єктів класу. В ідеальному випадку класи-домішки не мають власної ієрархії спадкування і не мають полів, а мають лише методи

1. **Яка роль функції super() при спадкуванні?**

Функція super() грає важливу роль при спадкуванні в об'єктно-орієнтованому програмуванні. Вона дозволяє виконувати методи батьківського класу в контексті підкласу.

**Висновок:** Я оволодів навиками реалізації парадигм об’єктно-орієнтованого програмування використовуючи засоби мови Python.