Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Кафедра «Електронних обчислювальних машин»

Звіт
з лабораторної роботи № 9
з дисципліни: «Кросплатформенні засоби програмування»
на тему: «ОСНОВИ ОБ’ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ У PYTHON»

Виконав:
студент групи КІ-306
Хмільовський С. Р.
Прийняв:
доцент кафедри ЕОМ
Іванов Ю. С.

Львів – 2023

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

Завдання (варіант № 22)

1. Написати та налагодити програму на мові Python згідно варіанту **(Базовий клас “Автомат” → Похідний клас “Штурмова гвинтівка”).** 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.

Вихідний код програми:

Файл AssaultRifle.py

```
#Базовий клас "Автомат":
class AssaultRifle:
    def __init__(self, model, ammunition, caliber):
        self.model = model
        self.ammunition = ammunition
        self.caliber = caliber

    def reload(self, rounds):
        if rounds > 0:
            self.ammunition += rounds
            return f"Reloaded with {rounds} rounds. \nCurrent ammunition count:
{self.ammunition}"
        else:
            return"Invalid number of rounds. Must be greater than zero."

    def burst_fire(self, bursts):
        bullets_to_fire = bursts * 3
        if self.ammunition >= bullets_to_fire:
            self.ammunition -= bullets_to_fire
```

```
            for i in range(bursts):
                print("Burst fire! (3 bullets used)")
            return f"Remaining ammunition: {self.ammunition}"
        else:
            return "Not enough ammunition to fire."

    def automatic_fire(self, bullets):
        if self.ammunition >= bullets:
            self.ammunition -= bullets
            for i in range(bullets):
                print(f"Automatic Fire! (Bullet used: {i + 1})")
            return f"Remaining ammunition: {self.ammunition}"
        else:
            return "Not enough ammunition to fire."

    def single_fire(self, bullets):
        bullets_to_fire = bullets
        if self.ammunition >= bullets_to_fire:
            self.ammunition -= bullets_to_fire
            for i in range(bullets):
                print("Single fire! (1 bullet used)")
            return f"Remaining ammunition: {self.ammunition}"
        else:
            return "Not enough ammunition to fire."

    def display_info(self):
        return f"Model: {self.model}\nCaliber: {self.caliber}\nAmmunition:
{self.ammunition}"
```

Файл CustomAssaultRifle.py

```
#Похідний клас "Штурмова гвинтівка":

from AssaultRifle import AssaultRifle

class CustomAssaultRifle(AssaultRifle):
    def __init__(self, model, ammunition, caliber, price, weight, camouflage,
customization):
        super().__init__(model, ammunition, caliber)
        # Додаткові властивості для похідного класу
        self.price = price
        self.weight = weight
        self.camouflage = camouflage
        self.customization = customization

    def customize(self, new_features):
        self.customization = new_features
        return f"Customization added: {self.customization}"

    def change_caliber(self, new_caliber):
        self.caliber = new_caliber
        return f"Caliber changed to: {self.caliber} mm"

    def calculate_total_price(self, custom_price):
        total_price = custom_price + self.price
        return f"Total price (including customization): {total_price} $"

    def change_weight(self, change):
        self.weight += change
        return f"Total weight (including customization): {self.weight} kg"

    def change_camouflage(self, new_camouflage):
        self.camouflage = new_camouflage
        return f"Camouflage changed to: {self.camouflage}"
```

```
    def display_info(self):
        base_info = super().display_info()
        return f"{base_info}\nPrice: ${self.price}\nWeight: {self.weight}
kg\nCamouflage: {self.camouflage}\nCustomization: {self.customization}"
```

Файл Main.py

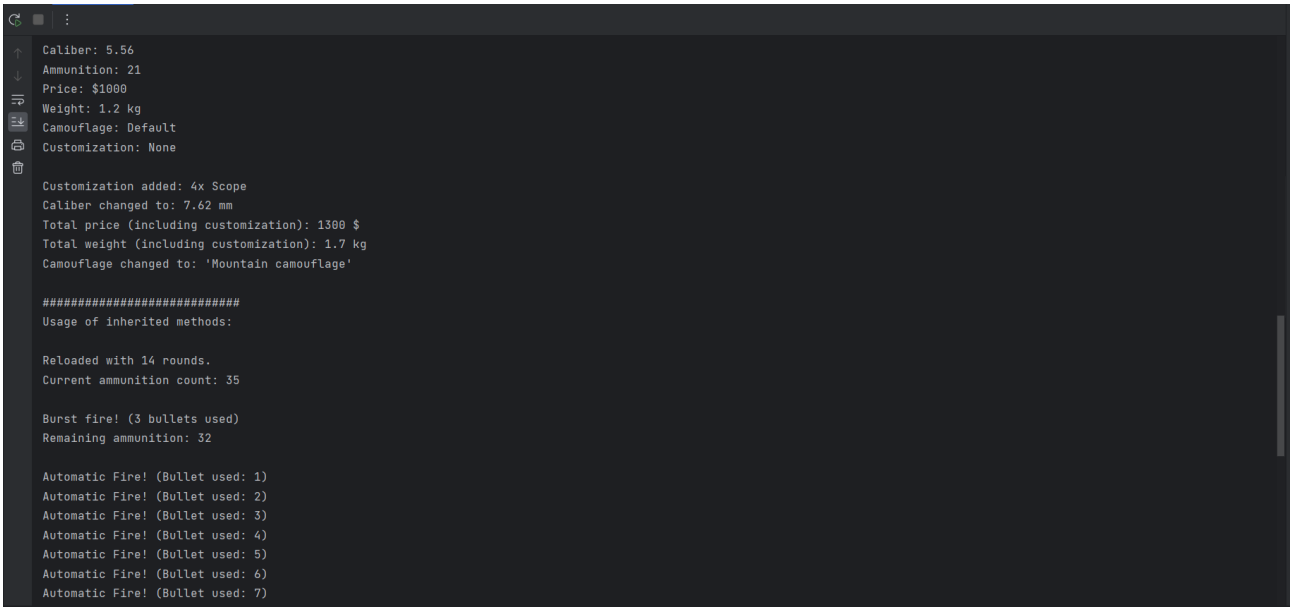
```
#Main функція програми:

from AssaultRifle import AssaultRifle
from CustomAssaultRifle import CustomAssaultRifle

def main():
    rifle1 = AssaultRifle("AR-15", 30, "5.56")

    print("\nAssaultRifle Info:")
    print(rifle1.display_info())
    print()

    #Використання методу перезарядки
    reload_message = rifle1.reload(20)
    print(reload_message)

    # Використання методу стрільби по три патрони
    print()
    burst_fire_message = rifle1.burst_fire(3)
    print(burst_fire_message)

    # Використання методу автоматичної стрільби
    print()
    automatic_fire_message = rifle1.automatic_fire(11)
    print(automatic_fire_message)

    # Використання методу одиночної стрільби
    print()
    single_fire_message = rifle1.single_fire(25)
    print(single_fire_message)


    custom_rifle1 = CustomAssaultRifle("CustomAR-15", 21, "5.56", 1000, 1.2,
"Default", "None")
    print()
    print("\nCustomAssaultRifle Info:")
    print(custom_rifle1.display_info())

    print()
    print(custom_rifle1.customize('4x Scope'))
    print(custom_rifle1.change_caliber("7.62"))
    print(custom_rifle1.calculate_total_price(300))
    print(custom_rifle1.change_weight(0.5))
    print(custom_rifle1.change_camouflage("\'Mountain camouflage\'"))

    print("\n############################")
    print("Usage of inherited methods: \n")
    # Використання методу перезарядки
    reload_message = custom_rifle1.reload(14)
    print(reload_message)

    # Використання методу стрільби по три патрони
    print()
    burst_fire_message = custom_rifle1.burst_fire(1)
    print(burst_fire_message)

    # Використання методу автоматичної стрільби
    print()
```

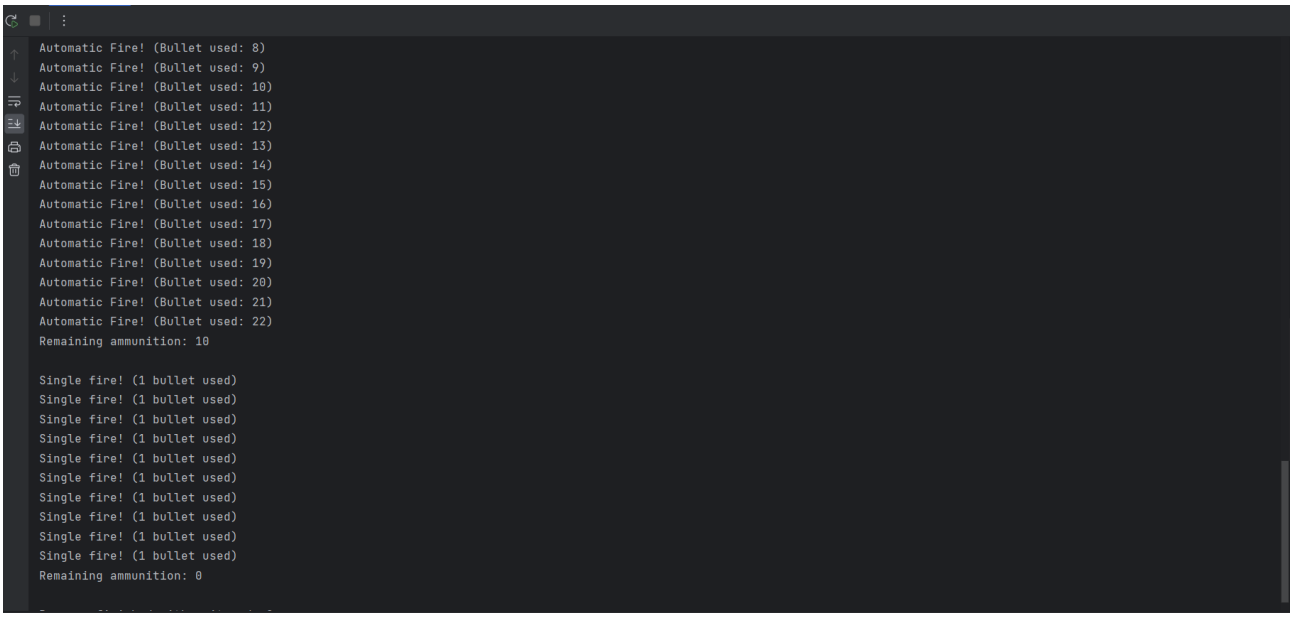
```
    automatic_fire_message = custom_rifle1.automatic_fire(22)
    print(automatic_fire_message)

    # Використання методу одиночної стрільби
    print()
    single_fire_message = custom_rifle1.single_fire(10)
    print(single_fire_message)


if __name__ == "__main__":
    main()
```

Результат виконання програми:

```
D:\Java[CPPT]\LAB9\Scripts\python.exe "D:\Java[CPPT]\LAB9\Main.py"

AssaultRifle Info:
Model: AR-15
Caliber: 5.56
Ammunition: 30

Reloaded with 20 rounds.
Current ammunition count: 50

Burst fire! (3 bullets used)
Burst fire! (3 bullets used)
Burst fire! (3 bullets used)
Remaining ammunition: 41

Automatic Fire! (Bullet used: 1)
Automatic Fire! (Bullet used: 2)
Automatic Fire! (Bullet used: 3)
Automatic Fire! (Bullet used: 4)
Automatic Fire! (Bullet used: 5)
Automatic Fire! (Bullet used: 6)
Automatic Fire! (Bullet used: 7)
Automatic Fire! (Bullet used: 8)
Automatic Fire! (Bullet used: 9)
Automatic Fire! (Bullet used: 10)
Automatic Fire! (Bullet used: 11)
Remaining ammunition: 30

Single fire! (1 bullet used)
```

```
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Single fire! (1 bullet used)
Remaining ammunition: 5


CustomAssaultRifle Info:
Model: CustomAR-15
```





Відповіді на контрольні запитання:

1. Що таке модулі?

Модулі в Python - це файли, які містять код Python. Вони використовуються для організації коду на логічні блоки і полегшення управління.

2. Як імпортувати модуль?

Модуль можна імпортувати за допомогою ключового слова import. Наприклад:

```
import my_module
```

3. Як оголосити клас?

Клас оголошується за допомогою ключового слова class. Наприклад:

```
class MyClass:
    # тіло класу
```

4. Що може міститися у класі?

У класі можуть міститися атрибути (змінні) і методи (функції), що дозволяє організувати дані та пов'язані з ними операції.

5. Як називається конструктор класу?

Конструктор класу називається `__init__`. Це спеціальний метод, який викликається автоматично при створенні нового об'єкта класу.

6. Як здійснити спадкування?

Спадкування здійснюється шляхом вказання батьківського класу у визначенні нового класу. Наприклад:

```
class ChildClass(ParentClass):
    # тіло класу
```

7. Які види спадкування існують?

У Python існують єдине спадкування, коли клас може успадковувати властивості лише від одного батьківського класу.

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

Небезпека множинного спадкування полягає в можливому конфлікті імен та складнощах з розумінням коду. Для уникнення цього слід стежити за чітким розподілом відповідальностей і уникати зайвого спадкування.

9. Що таке класи-домішки?

Класи-домішки в Python - це класи, які містять функціональність для багатьох класів. Об'єкти цих класів можуть включати в себе функціональність, яка необов'язково визначається в класах, які успадковуються від них.

10.Яка роль функції super() при спадкуванні?

Функція `super()` використовується для отримання доступу до методів батьківського класу. Вона дозволяє вам викликати методи батьківського класу, щоб доповнити або замінити їх у підкласі.

Висновок:

На даній лабораторній роботі я оволодів навичками використання засобів об'єктно-орієнтованого програмування мовою Python. Також ознайомився з ключовими аспектами цієї парадигми, включаючи створення та використання класів, роботу з об'єктами, та використання спадкування для покращення ефективності програми.