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

Чаус Б.В.

**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 № 24)**

**24. Спорядження військового альпініста**

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.

**Вихідний код програми**

Файл AlpinistEquipment.py

class AlpinistEquipment:

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

# Initialize dictionaries to keep track of quantities and weights for items

self.quantities = {}

self.weights = {}

# Initialize an array to store item names, with initial size of 10

self.item\_names = [None] \* 10

# Initialize item count to keep track of how many items are in the inventory

self.item\_count = 0

def resize\_item\_names\_array(self):

# Resize the item names array by doubling its size and copying existing items

new\_array = [None] \* (len(self.item\_names) \* 2)

new\_array[:len(self.item\_names)] = self.item\_names

self.item\_names = new\_array

def add\_item(self, item\_name, quantity, weight):

# Add or update item quantity and weight in the respective dictionaries

if item\_name in self.quantities:

self.quantities[item\_name] += quantity

else:

self.quantities[item\_name] = quantity

self.weights[item\_name] = weight

# Resize item names array if it's full

if self.item\_count == len(self.item\_names):

self.resize\_item\_names\_array()

# Log the action of adding an item

self.write\_to\_log\_file(f"Added {quantity} {item\_name}(s) with a total weight of {quantity \* weight} kg.")

def remove\_item(self, item\_name, quantity):

# Remove a specified quantity of an item from the inventory

if item\_name in self.quantities:

current\_quantity = self.quantities[item\_name]

if current\_quantity >= quantity:

self.quantities[item\_name] -= quantity

# Log the action of removing an item

self.write\_to\_log\_file(f"Removed {quantity} {item\_name}(s).")

else:

print(f"Error: Not enough {item\_name} in inventory.")

else:

print(f"Error: {item\_name} not found in inventory.")

def get\_total\_weight(self):

# Calculate and return the total weight of all items in the inventory

total\_weight = 0

for item\_name in self.quantities:

total\_weight += self.quantities[item\_name] \* self.weights[item\_name]

return total\_weight

def display\_inventory(self):

# Display the inventory, including item names, quantities, and weights

print("Inventory:")

for item\_name in self.quantities:

print(f"Item Name: {item\_name}")

print(f"Quantity: {self.quantities[item\_name]}")

print(f"Weight: {self.weights[item\_name]} kg")

def display\_all\_inventory\_names(self):

# Display all item names in the inventory

for i in range(self.item\_count):

print(self.item\_names[i])

def get\_quantity(self, item\_name):

# Get the quantity of a specific item in the inventory

return self.quantities.get(item\_name, 0)

def update\_item(self, item\_name, new\_quantity, new\_weight):

# Update the quantity and weight of a specific item in the inventory

self.quantities[item\_name] = new\_quantity

self.weights[item\_name] = new\_weight

# Log the action of updating an item

self.write\_to\_log\_file(f"Updated {item\_name} to {new\_quantity} quantity with a weight of {new\_weight} kg.")

def remove\_all\_items(self):

# Remove all items from the inventory

self.quantities.clear()

self.weights.clear()

# Log the action of removing all items

self.write\_to\_log\_file("All items removed from inventory.")

def contains\_item(self, item\_name):

# Check if a specific item is in the inventory

return item\_name in self.quantities

def clear\_log\_file(self):

# Clear the content of the log file

with open("log.txt", "w"):

pass

def write\_to\_log\_file(self, message):

# Write a message to the log file

with open("log.txt", "a") as file:

file.write(message + "\n")

Файл MilitaryAlpinistEquipment.py

from AlpinistEquipment import AlpinistEquipment

class MilitaryAlpinistEquipment(AlpinistEquipment):

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

# Call the constructor of the parent class AlpinistEquipment

super().\_\_init\_\_()

# Initialize an additional attribute specific to MilitaryAlpinistEquipment

self.night\_vision\_enabled = False

def enable\_night\_vision(self):

# Enable night vision

self.night\_vision\_enabled = True

def disable\_night\_vision(self):

# Disable night vision

self.night\_vision\_enabled = False

def is\_night\_vision\_enabled(self):

# Check if night vision is enabled

return self.night\_vision\_enabled

def perform\_military\_action(self):

# Perform a military alpinist action

print("Performing military alpinist action...")

def is\_combat\_ready(self):

# Check if the equipment is combat ready, based on night vision availability

return self.is\_night\_vision\_enabled()

Файл MilitaryAlpinistEquipmentApp.java

**package** KI.Chaus.Lab3;

**public** **class** MilitaryAlpinistEquipmentApp {

**public** **static** **void** main(String[] args) {

MilitaryAlpinistEquipment militaryEquipment = **new** MilitaryAlpinistEquipment();

militaryEquipment.clearLogFile();

militaryEquipment.addItem("Climbing Rope", 2, 3.5);

militaryEquipment.addItem("Carabiner", 10, 0.15);

militaryEquipment.updateItem("Climbing Rope", 3, 4.0);

militaryEquipment.displayInventory();

System.***out***.println("Total Weight: " + militaryEquipment.getTotalWeight() + " kg");

System.***out***.println("Quantity of Climbing Rope: " + militaryEquipment.getQuantity("Climbing Rope"));

**if** (militaryEquipment.containsItem("Helmet")) {

System.***out***.println("Helmet is in inventory.");

} **else** {

System.***out***.println("Helmet is not in inventory.");

}

militaryEquipment.removeItem("Climbing Rope", 2);

militaryEquipment.displayInventory();

System.***out***.println("Total Weight: " + militaryEquipment.getTotalWeight() + " kg");

militaryEquipment.removeAllItems()

militaryEquipment.enableNightVision();

militaryEquipment.performMilitaryAction();

}

}

Файл MilitaryAlpinistEquipmentApp.py

from MilitaryAlpinistEquipment import MilitaryAlpinistEquipment

def main():

# Create an instance of MilitaryAlpinistEquipment

military\_equipment = MilitaryAlpinistEquipment()

# Clear the log file

military\_equipment.clear\_log\_file()

# Add items to the inventory

military\_equipment.add\_item("Climbing Rope", 2, 3.5)

military\_equipment.add\_item("Carabiner", 10, 0.15)

# Update the quantity and weight of an item

military\_equipment.update\_item("Climbing Rope", 3, 4.0)

# Display the inventory and total weight

military\_equipment.display\_inventory()

print("Total Weight:", military\_equipment.get\_total\_weight(), "kg")

# Get the quantity of a specific item

print("Quantity of Climbing Rope:", military\_equipment.get\_quantity("Climbing Rope"))

# Check if an item is in the inventory

if military\_equipment.contains\_item("Helmet"):

print("Helmet is in inventory.")

else:

print("Helmet is not in inventory.")

# Remove a specified quantity of an item

military\_equipment.remove\_item("Climbing Rope", 2)

military\_equipment.display\_inventory()

print("Total Weight:", military\_equipment.get\_total\_weight(), "kg")

# Remove all items from the inventory

military\_equipment.remove\_all\_items()

# Enable night vision and perform a military action

military\_equipment.enable\_night\_vision()

military\_equipment.perform\_military\_action()

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

main()

Файл MilitaryAlpinistEquipmentInterface.py

class MilitaryAlpinistEquipmentInterface:

def perform\_military\_action(self):

# This method is intended to define the action a military alpinist equipment can perform.

# It should be overridden in subclasses that implement this interface.

pass

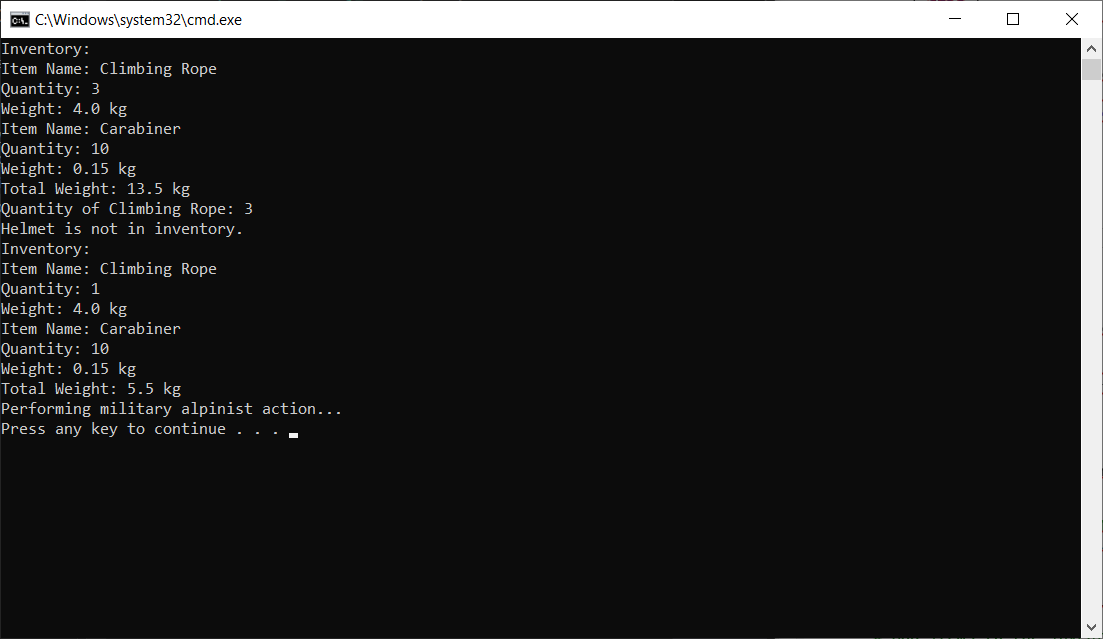
def is\_combat\_ready(self):

# This method is intended to determine if the military alpinist equipment is combat ready.

# It should be overridden in subclasses that implement this interface.

pass

**Результат виконання програми**



**Відповіді на контрольні запитання**

1. **Модулі** - це файли в Python, які містять пайтонівський код. Модулі дозволяють організувати код в логічні блоки та використовувати його в інших програмах.
2. **Імпортувати модуль** можна за допомогою ключового слова **import**. Наприклад, **import math** імпортує модуль **math**.
3. **Оголошення класу** починається з ключового слова **class**, за яким слідує ім'я класу і двокрапка. Наприклад, **class MyClass:**.
4. У **класі** можуть міститися:
   * **Атрибути**: Змінні, які присвоюються об'єктам класу.
   * **Методи**: Функції, що визначають поведінку об'єктів класу.
   * **Конструктор**: Спеціальний метод **\_\_init\_\_**, який викликається при створенні нового об'єкта.
   * **Інші спеціальні методи**: Наприклад, **\_\_str\_\_** для представлення об'єкта у вигляді рядка.
5. **Конструктор класу** має ім'я **\_\_init\_\_** і викликається автоматично при створенні нового об'єкта. В ньому встановлюються початкові значення атрибутів.
6. **Спадкування** в Python означає отримання властивостей та методів від батьківського класу. Це реалізується шляхом вказання батьківського класу у визначенні дочірнього класу.
7. **Види спадкування**:
   * **Одиночне спадкування**: Клас успадковує властивості лише від одного батьківського класу.
   * **Множинне спадкування**: Клас успадковує властивості від багатьох батьківських класів.
8. **Небезпеки при множинному спадкуванні**:
   * **Конфлікти імен**: Можуть виникнути колізії імен методів або атрибутів між батьківськими класами.
   * **Складність розуміння та утримання**: Множинне спадкування може зробити код складнішим для розуміння та утримання.
9. **Класи-домішки** (Mixin classes) - це спеціальні класи, які призначені для розширення функціональності інших класів шляхом надання додаткових методів та атрибутів.
10. **Функція super() при спадкуванні** використовується для виклику методів батьківського класу в дочірньому класі. Вона дозволяє уникнути проблем з однойменними методами в дочірньому та батьківському класах.

**Висновок**

Під час лабораторної роботи, я оволодів навиками реалізації парадигм об’єктно-орієнтованого програмування використовуючи засоби мови Python.