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();
    }
}
```

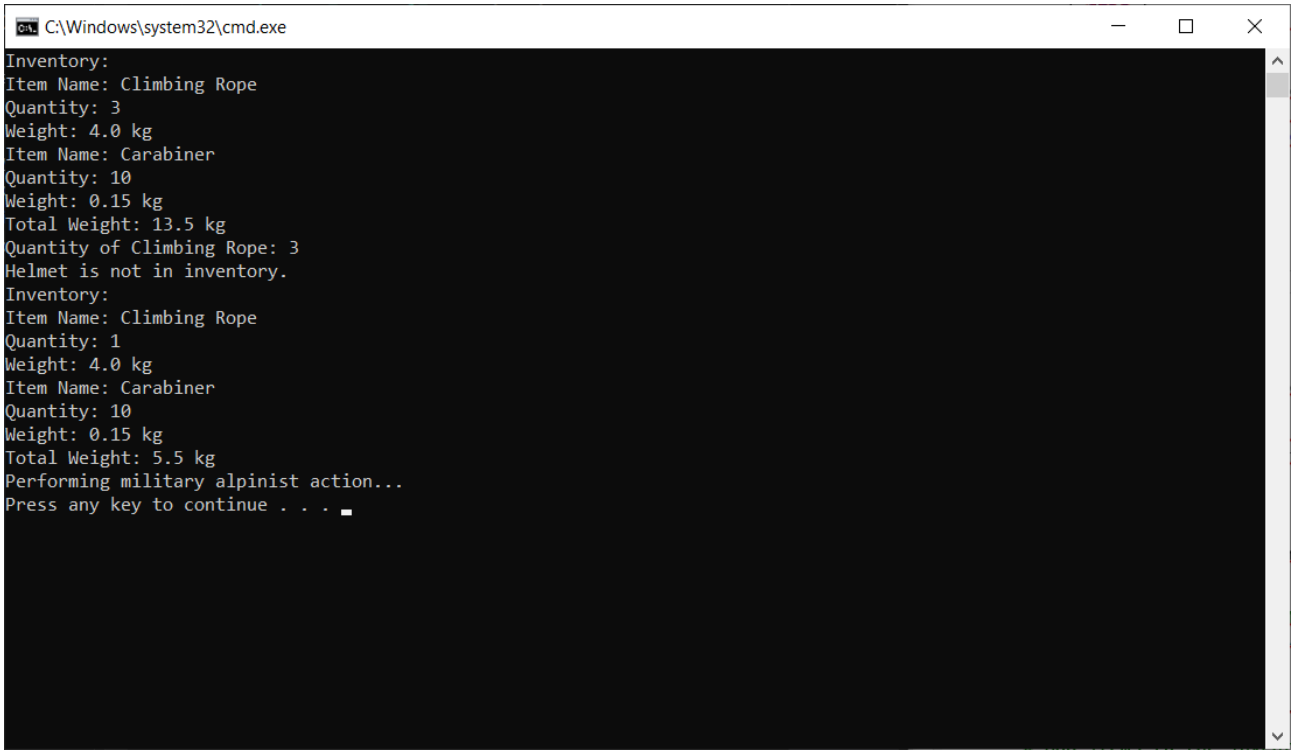
Файл MilitaryAlpinistEquipmentApp.py

```
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.