Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра «Електронних обчислювальних машин»



Звіт

з лабораторної роботи № 3

з дисципліни: «Кросплатформенні засоби програмування»

на тему: «СПАДКУВАННЯ ТА ІНТЕРФЕЙСИ»

**Виконав:**

студент групи КІ-306

Чаус Б.В.

**Прийняв:**

доцент кафедри ЕОМ

Іванов Ю. С.

Львів – 2023

**Мета роботи:** ознайомитися з спадкуванням та інтерфейсами у мові Java.

**Завдання (варіант № 24)**

**24. Спорядження військового альпініста**

1. Написати та налагодити програму на мові Java, що розширює клас, що реалізований у лабораторній роботі No2, для реалізації предметної області заданої варіантом.

Суперклас, що реалізований у лабораторній роботі No2, зробити абстрактним.

Розроблений підклас має забезпечувати механізми свого коректного функціонування та реалізовувати мінімум один інтерфейс. Програма має розміщуватися в пакеті Група.Прізвище.Lab3 та володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.

2. Автоматично згенерувати документацію до розробленого пакету.

3. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.

4. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації та завантажити його у ВНС.

5. Дати відповідь на контрольні запитання.

**Вихідний код програми**

Файл AlpinistEquipment.java

package KI.Chaus.Lab3;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.PrintWriter;

import java.util.HashMap;

import java.util.Map;

/\*\*

\* This class represents an inventory system for alpinist equipment.

\* It allows adding, removing, updating, and displaying items in the inventory.

\* It also keeps a log of all operations in a file named "log.txt".

\*

\* @author Chaus Bohdan

\* @version 1.0

\*/

public abstract class AlpinistEquipment {

private Map<String, Integer> quantities;

private Map<String, Double> weights;

/\*\*

\* Constructor to initialize the equipment inventory.

\*/

public AlpinistEquipment() {

quantities = new HashMap<>();

weights = new HashMap<>();

}

/\*\*

\* Adds a specified quantity of an item with its weight to the inventory.

\*

\* @param itemName The name of the item.

\* @param quantity The quantity of the item to be added.

\* @param weight The weight of a single item in kilograms.

\*/

public void addItem(String itemName, int quantity, double weight) {

quantities.put(itemName, quantities.getOrDefault(itemName, 0) + quantity);

weights.put(itemName, weight);

writeToLogFile("Added " + quantity + " " + itemName + "(s) with a total weight of " + (quantity \* weight) + " kg.");

}

/\*\*

\* Removes a specified quantity of an item from the inventory.

\*

\* @param itemName The name of the item.

\* @param quantity The quantity of the item to be removed.

\*/

public void removeItem(String itemName, int quantity) {

if (quantities.containsKey(itemName)) {

int currentQuantity = quantities.get(itemName);

if (currentQuantity >= quantity) {

quantities.put(itemName, currentQuantity - quantity);

writeToLogFile("Removed " + quantity + " " + itemName + "(s).");

} else {

System.out.println("Error: Not enough " + itemName + " in inventory.");

}

} else {

System.out.println("Error: " + itemName + " not found in inventory.");

}

}

/\*\*

\* Calculates and returns the total weight of all items in the inventory.

\*

\* @return The total weight of all items in kilograms.

\*/

public double getTotalWeight() {

double totalWeight = 0;

for (String itemName : quantities.keySet()) {

totalWeight += quantities.get(itemName) \* weights.get(itemName);

}

return totalWeight;

}

/\*\*

\* Displays the current inventory, including item names, quantities, and weights.

\*/

public void displayInventory() {

System.out.println("Inventory:");

for (String itemName : quantities.keySet()) {

System.out.println("Item Name: " + itemName);

System.out.println("Quantity: " + quantities.get(itemName));

System.out.println("Weight: " + weights.get(itemName) + " kg");

}

}

/\*\*

\* Gets the quantity of a specified item in the inventory.

\*

\* @param itemName The name of the item.

\* @return The quantity of the item, or 0 if not found.

\*/

public int getQuantity(String itemName) {

return quantities.getOrDefault(itemName, 0);

}

/\*\*

\* Updates the quantity and weight of a specified item in the inventory.

\*

\* @param itemName The name of the item.

\* @param newQuantity The new quantity of the item.

\* @param newWeight The new weight of a single item in kilograms.

\*/

public void updateItem(String itemName, int newQuantity, double newWeight) {

quantities.put(itemName, newQuantity);

weights.put(itemName, newWeight);

writeToLogFile("Updated " + itemName + " to " + newQuantity + " quantity with a weight of " + newWeight + " kg.");

}

/\*\*

\* Removes all items from the inventory.

\*/

public void removeAllItems() {

quantities.clear();

weights.clear();

writeToLogFile("All items removed from inventory.");

}

/\*\*

\* Checks if a specified item is present in the inventory.

\*

\* @param itemName The name of the item.

\* @return true if the item is present, false otherwise.

\*/

public boolean containsItem(String itemName) {

return quantities.containsKey(itemName);

}

/\*\*

\* Clears the contents of the log file.

\*/

public void clearLogFile() {

File logFile = new File("log.txt");

try {

PrintWriter writer = new PrintWriter(logFile);

writer.close();

} catch (FileNotFoundException e) {

e.printStackTrace();

}

}

/\*\*

\* Writes a message to the log file.

\*

\* @param message The message to be written to the log file.

\*/

private void writeToLogFile(String message) {

try (PrintWriter writer = new PrintWriter(new FileOutputStream(new File("log.txt"), true))) {

writer.println(message);

} catch (FileNotFoundException e) {

e.printStackTrace();

}

}

}

Файл MilitaryAlpinistEquipment.java

**package** KI.Chaus.Lab3;

/\*\*

\* This class represents military alpinist equipment and extends the AlpinistEquipment class.

\* It provides mechanisms for correct functioning specific to military equipment.

\* It also implements the MilitaryEquipment interface.

\*

\* **@author** Chaus Bohdan

\* **@version** 1.0

\*/

**public** **class** MilitaryAlpinistEquipment **extends** AlpinistEquipment **implements** MilitaryAlpinistEquipmentInterface {

// Indicates whether night vision mode is enabled.

**private** **boolean** nightVisionEnabled;

// Constructor initializes nightVisionEnabled to false by default.

**public** MilitaryAlpinistEquipment() {

nightVisionEnabled = **false**;

}

/\*\*

\* Enables night vision mode.

\*/

**public** **void** enableNightVision() {

nightVisionEnabled = **true**;

}

/\*\*

\* Disables night vision mode.

\*/

**public** **void** disableNightVision() {

nightVisionEnabled = **false**;

}

/\*\*

\* Checks if night vision mode is enabled.

\*

\* **@return** true if night vision is enabled, false otherwise.

\*/

**public** **boolean** isNightVisionEnabled() {

**return** nightVisionEnabled;

}

/\*\*

\* Overrides the method in the interface.

\* Prints a message indicating a military alpinist action is being performed.

\*/

@Override

**public** **void** performMilitaryAction() {

System.***out***.println("Performing military alpinist action...");

}

/\*\*

\* Overrides the method in the interface.

\* Indicates if the equipment is combat-ready based on night vision status.

\*

\* **@return** true if night vision is enabled, false otherwise.

\*/

@Override

**public** **boolean** isCombatReady() {

**return** isNightVisionEnabled();

}

}

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.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 MilitaryAlpinistEquipmentInterface.java

**package** KI.Chaus.Lab3;

/\*\*

\* This interface defines methods for military equipment.

\* Implementing classes must provide functionality for performing military actions.

\*

\* **@author** Chaus Bohdan

\* **@version** 1.0

\*/

**public** **interface** MilitaryAlpinistEquipmentInterface {

/\*\*

\* Performs a specific military action.

\*/

**void** performMilitaryAction();

/\*\*

\* Checks if the equipment is ready for combat.

\*

\* **@return** true if the equipment is combat-ready, false otherwise.

\*/

**boolean** isCombatReady();

}

**Результат виконання програми**

Inventory:

Item Name: Climbing Rope

Quantity: 3

Weight: 4.0 kg

Item Name: Carabiner

Quantity: 10

Weight: 0.15 kg

Total Weight: 13.5 kg

Quantity of Climbing Rope: 3

Helmet is not in inventory.

Inventory:

Item Name: Climbing Rope

Quantity: 1

Weight: 4.0 kg

Item Name: Carabiner

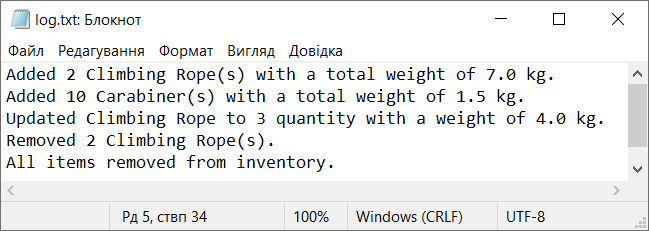
Quantity: 10

Weight: 0.15 kg

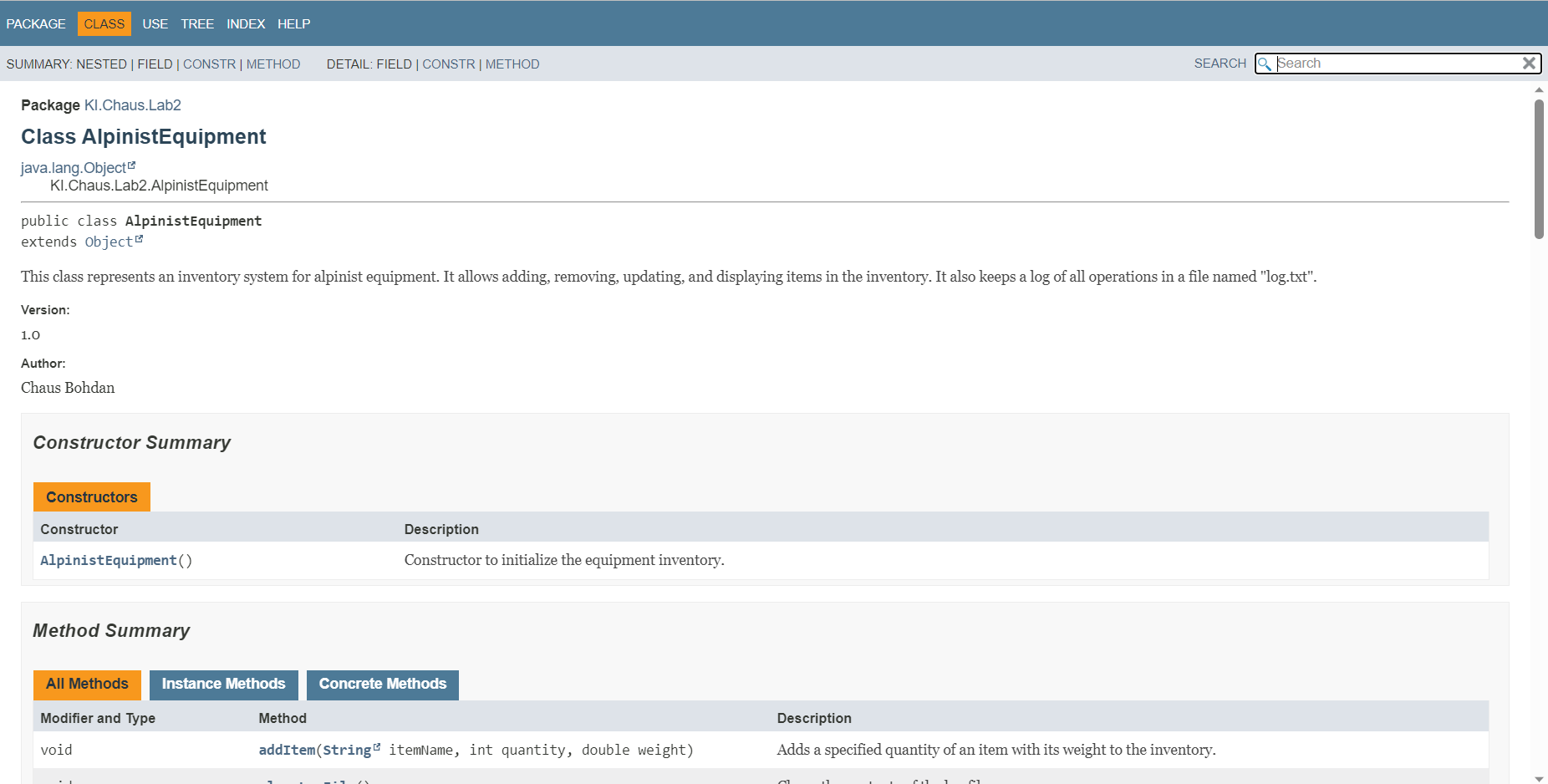
Total Weight: 5.5 kg

Performing military alpinist action...

**Текстовий файл з результатом виконання програми**



**Фрагмент згенерованої документації**



**Відповіді на контрольні запитання**

1. Синтаксис реалізації спадкування:

class Підклас extends Суперклас { // код }

1. Суперклас та підклас: Суперклас - це клас, від якого інший клас (підклас) успадковує властивості та методи.
2. Звернення до членів суперкласу з підкласу: Використовуючи ключове слово **super**, наприклад:

super.методСуперкласу();

1. Статичне зв’язування відбувається під час компіляції. Виклик методу вирішується на основі типу посилання, а не об’єкта.
2. Динамічне зв’язування відбувається під час виконання програми. Виклик методу вирішується на основі типу об’єкта, а не посилання.
3. Абстрактний клас - це клас, який не може мати екземплярів і може мати абстрактні методи. Його можна оголосити за допомогою ключового слова **abstract**.
4. Ключове слово **instanceof** використовується для перевірки того, чи об’єкт є екземпляром певного класу або його підкласу.
5. Для перевірки чи клас є підкласом іншого класу, можна використати **instanceof** або порівняти класи з допомогою **getClass()**.
6. Інтерфейс - це контракт, який визначає набір методів, але не надає реалізацію. В інтерфейсі всі методи за замовчуванням є абстрактними.
7. Щоб оголосити інтерфейс використовується ключове слово **interface**. Для його реалізації в класі використовується ключове слово **implements**. Наприклад:

interface Інтерфейс {

void метод();

}

class Клас implements Інтерфейс {

public void метод() {

// код

}

}

**Висновок**

Під час лабораторної роботи, я ознайомився з спадкуванням та інтерфейсами у мові Java.