

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*

Ярема Максим

Прийняв:

доцент кафедри ЕОМ

Іванов Ю. С.

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

Завдання (варіант № 29)

1. Написати та налагодити програму на мові Java, що розширює клас, що реалізований у лабораторній роботі №3, для реалізації предметної області заданої варіантом. Суперклас, що реалізований у лабораторній роботі №3, зробити абстрактним. Розроблений підклас має забезпечувати механізми свого коректного функціонування та реалізовувати мінімум один інтерфейс. Програма має розміщуватися в пакеті Група.Прізвище.Lab4 та володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.
2. Автоматично згенерувати документацію до розробленого пакету.
3. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації.
4. Дати відповідь на контрольні запитання

Вихідний код програми

Файл GasMaskApplication.java

```
package ki306.yarema.lab4;

import ki306.yarema.lab4.models.Filter;
import ki306.yarema.lab4.models.CommanderGasMask;

import java.io.IOException;

/**
 * Gas Mask Application class implements main method for CommanderGasMask
 * class possibilities demonstration
 *
 * @author Yarema Maksym
 * @version 1.0
 * @since version 1.0
 */
public class GasMaskApplication {
    /**
     * @param args
     */
    public static void main(String[] args) {
        CommanderGasMask commanderGasMask = new CommanderGasMask();

        commanderGasMask.checkStatus();

        commanderGasMask.breathe();
        commanderGasMask.breathe();

        commanderGasMask.sealMask();

        commanderGasMask.breathe();

        commanderGasMask.turnNightVisionOn();
    }
}
```

```

commanderGasMask.breathe();

commanderGasMask.replaceFilter(new Filter("A1B1E1K1"));

commanderGasMask.checkStatus();

commanderGasMask.sendSosSignal();

commanderGasMask.cleanAndReplaceFilter();

try {
commanderGasMask.dispose();
} catch (IOException e) {
e.printStackTrace();
}
}
}

```

Файл GasMask.java

```

package ki306.yarema.lab4.models;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
/**
 * Class Lab2YaremaKI306 implements laboratory work №4
 * An abstract class representing a gas mask and allowing various operations with it.
 * The gas mask has a type, a filter, a state (on or off), a usage count,
 * a sealing state, and a cleanliness state.
 *
 * @author Yarema Maksym
 * @version 1.2
 * @since version 1.0
 */
public abstract class GasMask {
    private final String type;
    private Filter filter;
    private boolean isNightVisionOn;
    private int usageCount;
    private boolean isSealed;
    private boolean isClean;
    private String logFileName;
    private FileWriter logFileWriter;

    /**
     * Constructor for creating a gas mask with default values.
     * Sets the mask type to "EN14387," the filter type to "A2B2E2K2," and other initial
     values.
     */
    public GasMask() {
        type = "EN14387";
        filter = new Filter("A2B2E2K2");
        isNightVisionOn = false;
        usageCount = 0;
        isSealed = false;
        isClean = true;
        logFileName = "GasMaskLog.txt";

        try {
            logFileWriter = new FileWriter(new File(logFileName));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Constructor for creating a gas mask with specified parameters.
     * @param maskType The type of the gas mask.
     * @param filterType The type of the filter.
     */
    public GasMask(String maskType, String filterType) {

```

```

type = maskType;
filter = new Filter(filterType);
isNightVisionOn = false;
usageCount = 0;
isSealed = false;
isClean = true;
logFileName = "GasMaskLog.txt";

try {
logFileWriter = new FileWriter(new File(logFileName));
} catch (IOException e) {
e.printStackTrace();
}
}

/**
 * Turn on night vision mode for the gas mask.
 */
public void turnNightVisionOn() {
isNightVisionOn = true;
log("Night vision mode is turned on.");
}

/**
 * Turn off night vision mode for the gas mask.
 */
public void turnNightVisionOff() {
isNightVisionOn = false;
log("Night vision mode is turned off.");
}

/**
 * Replace the filter of the gas mask with a new one.
 * @param newFilter The new filter.
 */
public void replaceFilter(Filter newFilter) {
filter = newFilter;
log("Filter replaced with a new one.");
}

/**
 * Breathe through the gas mask.
 */
public void breathe() {
if (isSealed && filter.isEffective()) {
incrementUsage();
log("Breathed safely using the gas mask.");
} else {
log("Gas mask is not sealed properly or the filter is not effective.");
}
}

/**
 * Seal the gas mask.
 */
public void sealMask() {
isSealed = true;
log("Gas mask is sealed.");
}

/**
 * Unseal the gas mask.
 */
public void unsealMask() {
isSealed = false;
log("Gas mask is unsealed.");
}

/**
 * Clean the gas mask.
 */
public void cleanMask() {
isClean = true;
log("Gas mask is cleaned.");
}

```

```

}

/**
 * Increment the usage count of the gas mask.
 */
private void incrementUsage() {
    usageCount++;
}

/**
 * Clean the gas mask and replace the filter with a new one.
 */
public void cleanAndReplaceFilter() {
    if (!isClean) {
        cleanMask();
    }
    replaceFilter(new Filter("A2B2E2K2"));
    log("Gas mask is cleaned and the filter is replaced.");
}

/**
 * Check the status of the gas mask and display status information.
 */
public void checkStatus() {
    String status = "Gas Mask Status:\n";
    status += "Type: " + type + "\n";
    status += "Filter Type: " + filter.getType() + "\n";
    status += "Is Night Vision On: " + isNightVisionOn + "\n";
    status += "Usage Count: " + usageCount + "\n";
    status += "Is Sealed: " + isSealed + "\n";
    status += "Is Clean: " + isClean + "\n";
    log(status);
}

/**
 * Return type of gas mask.
 */
public String getType() {
    return type;
}

/**
 * Write a message to the log file.
 * @param message The message to be logged.
 */
public void log(String message) {
    try {
        logFileWriter.write(message + "\n");
        logFileWriter.flush();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

/**
 * Dispose of the gas mask and close the log file.
 * @throws IOException Thrown in case of input/output errors.
 */
public void dispose() throws IOException {
    logFileWriter.close();
}
}

```

Файл Filter.java

```

package ki306.yarema.lab4.models;

/**
 * A class representing a filter used in a gas mask and allowing various operations with it.

```

```

* The filter has a type and an effectiveness state.
*
* @author Yarema Maksym
* @version 1.0
* @since version 1.0
*/
public class Filter {
    private final String type;
    private boolean isEffective;

    /**
     * Constructor for creating a filter with default values.
     * Sets the filter type to "A2B2E2K2" and initializes it as effective.
     */
    public Filter() {
        type = "A2B2E2K2";
        isEffective = true;
    }

    /**
     * Constructor for creating a filter with a specified type.
     * @param type The type of the filter.
     */
    public Filter(String type) {
        this.type = type;
        isEffective = true;
    }

    /**
     * Make the filter ineffective.
     */
    public void makeIneffective() {
        isEffective = false;
    }

    /**
     * Check if the filter is effective.
     * @return true if the filter is effective, false otherwise.
     */
    public boolean isEffective() {
        return isEffective;
    }

    /**
     * Get the type of the filter.
     * @return The type of the filter.
     */
    public String getType() {
        return type;
    }
}

```

Файл CommanderEquipment.java

```

package ki306.yarema.lab4.models;

/**
 * The CommanderEquipment interface represents the equipment used by the commander.
 * All classes that implement this interface must provide methods for getting the type of
 * equipment
 * and sending an SOS signal.
 */
public interface CommanderEquipment {
    /**
     * Returns the type of equipment of the commander".
     */
    String getEquipmentType();

    /**
     * Sends an SOS signal in case of an emergency.
     */
    void sendSosSignal();
}

```

Файл CommanderGasMask.java

```
package ki306.yarema.lab4.models;

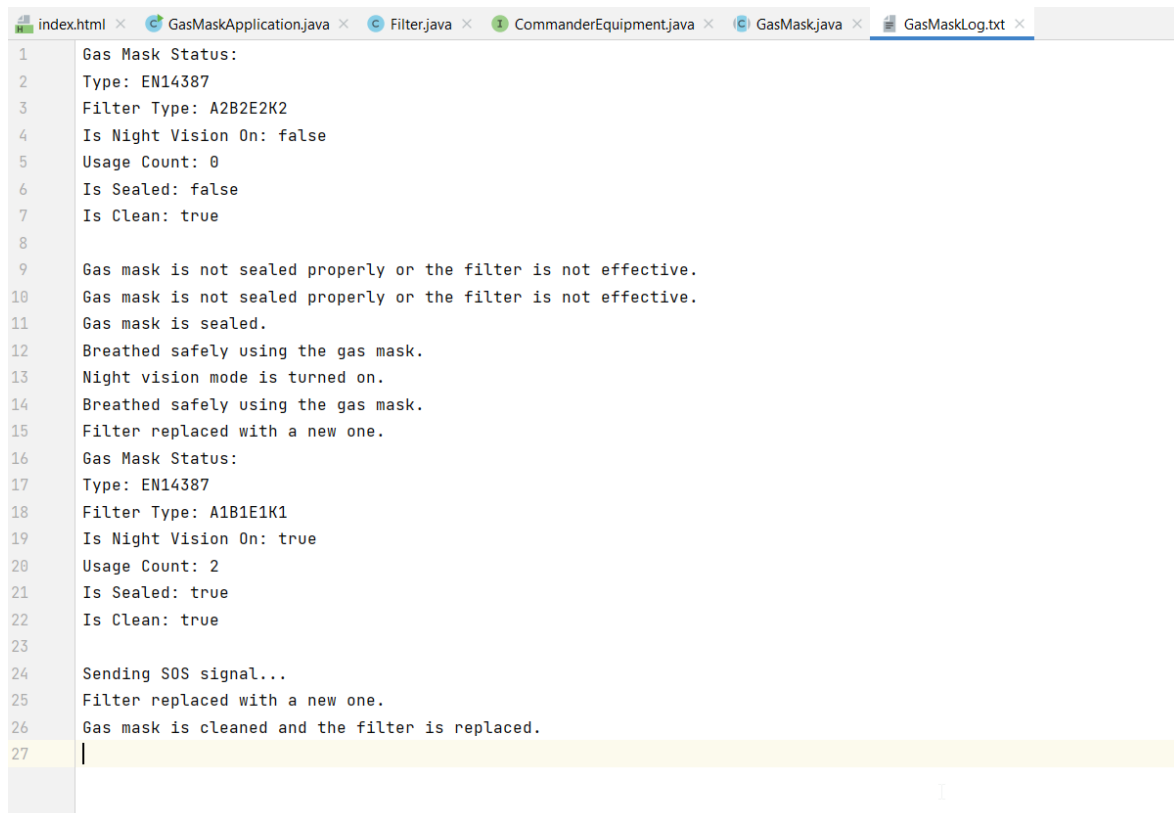
/**
 * The CommanderGasMask class represents the commander's gas mask,
 * which implements the CommanderEquipment interface and inherits the properties of the
 * GasMask class.
 *
 * @author Yarema Maksym
 * @version 1.0
 * @since version 1.0
 */
public class CommanderGasMask extends GasMask implements CommanderEquipment {

    /**
     * Returns the type of equipment of the commander - "Gas mask".
     *
     * @return type of commanders gas mask
     */
    @Override
    public String getEquipmentType() {
        return getType();
    }

    /**
     * Sends an SOS signal in case of an emergency.
     */
    @Override
    public void sendSosSignal() {
        log("Sending SOS signal...");
    }
}
```


Результат виконання програми

GasMaskLog.txt:



```
index.html x GasMaskApplication.java x Filter.java x CommanderEquipment.java x GasMask.java x GasMaskLog.txt x
1 Gas Mask Status:
2 Type: EN14387
3 Filter Type: A2B2E2K2
4 Is Night Vision On: false
5 Usage Count: 0
6 Is Sealed: false
7 Is Clean: true
8
9 Gas mask is not sealed properly or the filter is not effective.
10 Gas mask is not sealed properly or the filter is not effective.
11 Gas mask is sealed.
12 Breathed safely using the gas mask.
13 Night vision mode is turned on.
14 Breathed safely using the gas mask.
15 Filter replaced with a new one.
16 Gas Mask Status:
17 Type: EN14387
18 Filter Type: A1B1E1K1
19 Is Night Vision On: true
20 Usage Count: 2
21 Is Sealed: true
22 Is Clean: true
23
24 Sending SOS signal...
25 Filter replaced with a new one.
26 Gas mask is cleaned and the filter is replaced.
27 |
```

Фрагмент згенерованої документації



SEARCH:

Package ki306.yarema.lab4.models

package ki306.yarema.lab4.models

Related Packages

Package	Description
ki306.yarema.lab4	

All Classes and Interfaces **Interfaces** **Classes**

Class	Description
CommanderEquipment	The CommanderEquipment interface represents the equipment used by the commander.
CommanderGasMask	The CommanderGasMask class represents the commander's gas mask, which implements the CommanderEquipment interface and inherits the properties of the GasMask class.
Filter	A class representing a filter used in a gas mask and allowing various operations with it.
GasMask	Class Lab2YaremaKI306 implements laboratory work №4 An abstract class representing a gas mask and allowing various operations with it.

Відповіді на контрольні запитання

- Синтаксис реалізації спадкування.
 - ```
class МійКлас implements Інтерфейс {
 // тіло класу
}
```
- Що таке суперклас та підклас?
  - суперклас - це клас, від якого інший клас успадковує властивості та методи.
  - Підклас - це клас, який успадковує властивості та методи від суперкласу.
- Як звернутися до членів суперкласу з підкласу?
  - ```
супер.назваМетоду([параметри]); // виклик методу суперкласу
```
 - ```
супер.назваПоля; // звернення до поля суперкласу
```



4. Коли використовується статичне зв'язування при виклику методу?
  - Статичне зв'язування використовується, коли метод є приватним, статичним, фінальним або конструктором. В таких випадках вибір методу відбувається на етапі компіляції.
5. Як відбувається динамічне зв'язування при виклику методу?
  - вибір методу для виклику відбувається під час виконання програми на основі фактичного типу об'єкта.
6. Що таке абстрактний клас та як його реалізувати?
  - це клас, який має один або більше абстрактних методів (методів без реалізації). Щоб створити абстрактний клас, використовується ключове слово `abstract`.  
Приклад:

```
abstract class АбстрактнийКлас {
 abstract void абстрактнийМетод();
}
```
7. Для чого використовується ключове слово `instanceof`?
  - для перевірки, чи об'єкт належить до певного класу або інтерфейсу.  
Синтаксис:

```
if (об'єкт instanceof Клас) {
 // код, який виконується, якщо об'єкт належить до класу
}
```
8. Як перевірити чи клас є підкласом іншого класу?
  - В Java використовується ключове слово `extends`, щоб вказати, що клас є підкласом іншого класу. Перевірити, чи один клас є підкласом іншого класу можна шляхом аналізу ієрархії успадкування.
9. Що таке інтерфейс?
  - це абстрактний тип даних, який визначає набір методів, але не надає їх реалізацію. Всі методи інтерфейсу є загальнодоступними та автоматично є `public`. Інтерфейси використовуються для створення контрактів, які класи повинні реалізувати.
10. Як оголосити та застосувати інтерфейс?
  - Для оголошення інтерфейсу використовується ключове слово `interface`.  
Синтаксис:

```
interface Інтерфейс {
 // оголошення методів та констант
}
```
  - Для застосування інтерфейсу в класі використовується ключове слово `implements`.  
Синтаксис:

```
class МійКлас implements Інтерфейс {
 // реалізація методів інтерфейсу
}
```

## **Висновок**

У ході виконання даної лабораторної роботи, я отримав навички роботи з концепціями спадкування та інтерфейсами в мові програмування Java. Ознайомившись з цими важливими аспектами об'єктно-орієнтованого програмування, я зрозумів їх роль у створенні більш структурованих і гнучких програм.