

Міністерство освіти і науки України
Національний університет «Львівська політехніка»

Кафедра ЕОМ



Лабораторна робота №2

з дисципліни: «Кросплатформенні засоби програмування»

на тему: «Класи та пакети»

Варіант № 25

Виконав:

ст. гр. КІ-305

Федусь Н.В.

Прийняв:

Іванов Ю. С.

Львів – 2023

Мета: Ознайомитися з процесом розробки класів та пакетів мовою Java.

Завдання:

1. Написати та налагодити програму на мові Java, що реалізує у вигляді класу предметну область згідно варіанту. Програма має задовольняти наступним вимогам:

- програма має розміщуватися в пакеті Група.Прізвище.Lab2;
- клас має містити мінімум 3 поля, що є об'єктами класів, які описують складові частини предметної області;
- клас має містити кілька конструкторів та мінімум 10 методів;
- для тестування і демонстрації роботи розробленого класу розробити клас-драйвер;
- методи класу мають вести протокол своєї діяльності, що записується у файл;
- розробити механізм коректного завершення роботи з файлом (не надіятися на метод `finalize()`);
- програма має 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.

Код ConditionerApp.java:

```
package KI305.Fedus.Lab2;
```

```
import java.io.FileNotFoundException;
```

```
/**
```

```
 * ConditionerApp class with the entry point for the program.
```

```
 * Author: Fedus N.V. KI-305
```

```
 */
```

```
public class ConditionerApp {
```

```
/**
```

```
 * The main method is the entry point for the program.
```

```

* Creates instances of the Conditioner class, demonstrates its methods, and closes the logger files.
*
* @param args command-line arguments (not used in this program)
* @throws FileNotFoundException if the log file cannot be created.
*/
public static void main(String[] args) throws FileNotFoundException {
    // Create a default Conditioner instance
    Conditioner conditioner = new Conditioner();
    conditioner.getFactoryName(); // Demonstrate getFactoryName method
    conditioner.getConditionColor(); // Demonstrate getConditionColor method
    conditioner.getConditionMode(); // Demonstrate getConditionMode method
    conditioner.setLowCondition(); // Demonstrate setLowCondition method
    conditioner.getConditionMode();

    // Create a custom Conditioner instance with a specific name and color
    Conditioner conditioner1 = new Conditioner("Conditioner 2077",
        Conditioner.ConditionColor.YELLOW);
    conditioner1.setHighCondition(); // Set the conditioner to high mode
    System.out.println(conditioner1.getConditionMode()); // Print the current mode

    // Close logger files for both instances
    conditioner.closeLoggerFile();
    conditioner1.closeLoggerFile();
}
}

```

Код Conditioner.java:

```

package KI305.Fedus.Lab2;

import java.io.*;

/**
 * Conditioner class represents a basic air conditioner.
 */
public class Conditioner {
    // Data fields
    private String factoryName;
    private ConditionColor conditionColor;
    private ConditionMode conditionMode;
    private static int conditionerNumber = 1;

```

```
private PrintWriter outputStream = new PrintWriter(new File(String.format("ConditionerLogger
%s.txt", conditionerNumber)));
```

```
/**
```

```
* Default constructor for the Conditioner class.
* Initializes the conditioner with default values and increments the conditioner number.
* @throws FileNotFoundException if the log file cannot be created.
*/
```

```
public Conditioner() throws FileNotFoundException {
    factoryName = String.format("#%s Conditioner", conditionerNumber);
    conditionColor = ConditionColor.WHITE;
    conditionMode = ConditionMode.TURNED_OFF;
    outputStream.println("Creating a conditioner");
    ++conditionerNumber;
}
```

```
/**
```

```
* Constructor for the Conditioner class with a custom factory name.
* Initializes the conditioner with the specified factory name and default values.
* @param factoryName the custom factory name for the conditioner.
* @throws FileNotFoundException if the log file cannot be created.
*/
```

```
public Conditioner(String factoryName) throws FileNotFoundException {
    this.factoryName = factoryName;
    conditionColor = ConditionColor.WHITE;
    conditionMode = ConditionMode.TURNED_OFF;
    outputStream.println("Creating a conditioner");
    ++conditionerNumber;
}
```

```
/**
```

```
* Constructor for the Conditioner class with a custom factory name and condition color.
* Initializes the conditioner with the specified factory name, condition color, and default mode.
* @param factoryName the custom factory name for the conditioner.
* @param conditionColor the color of the conditioner.
* @throws FileNotFoundException if the log file cannot be created.
*/
```

```
public Conditioner(String factoryName, ConditionColor conditionColor) throws
FileNotFoundException {
    this.factoryName = factoryName;
    this.conditionColor = conditionColor;
    conditionMode = ConditionMode.TURNED_OFF;
```

```

    outputStream.println("Creating a conditioner");
    ++conditionerNumber;
}

// Enum representing different modes of the conditioner
private enum ConditionMode {
    TURNED_OFF, LOW, MEDIUM, HIGH
}

// Enum representing different colors of the conditioner
public enum ConditionColor {
    WHITE, BLACK, RED, PINK, YELLOW, GREEN, BLUE
}

// Getter methods

/**
 * Get the factory name of the conditioner.
 * @return the factory name of the conditioner.
 */
public String getFactoryName() {
    outputStream.println("getFactoryName: " + factoryName);
    return factoryName;
}

/**
 * Get the current mode of the conditioner.
 * @return the current mode of the conditioner.
 */
public ConditionMode getConditionMode() {
    outputStream.println("getConditionMode: " + conditionMode);
    return conditionMode;
}

/**
 * Get the color of the conditioner.
 * @return the color of the conditioner.
 */
public ConditionColor getConditionColor() {
    outputStream.println("getConditionColor: " + conditionColor);
    return conditionColor;
}

```

```
// Setter methods
```

```
/**
```

```
 * Turn off the conditioner if it is not already turned off.
```

```
 */
```

```
public void turnOffCondition() {  
    if (conditionMode == ConditionMode.TURNED_OFF) {  
        System.out.println("Condition is already turned off.");  
        outputStream.println("Condition is already turned off.");  
    } else {  
        System.out.println("Condition is turned off.");  
        outputStream.println("Condition is turned off.");  
        conditionMode = ConditionMode.TURNED_OFF;  
    }  
}
```

```
/**
```

```
 * Set the conditioner to low mode if it is not already in low mode.
```

```
 */
```

```
public void setLowCondition() {  
    if (conditionMode == ConditionMode.LOW) {  
        System.out.println("Condition is already in low mode.");  
        outputStream.println("Condition is already in low mode.");  
    } else {  
        System.out.println("Condition is set in low mode.");  
        outputStream.println("Condition is set in low mode.");  
        conditionMode = ConditionMode.LOW;  
    }  
}
```

```
/**
```

```
 * Set the conditioner to medium mode if it is not already in medium mode.
```

```
 */
```

```
public void setMediumCondition() {  
    if (conditionMode == ConditionMode.MEDIUM) {  
        System.out.println("Condition is already in medium mode.");  
        outputStream.println("Condition is already in medium mode.");  
    } else {  
        System.out.println("Condition is set in medium mode.");  
        outputStream.println("Condition is set in medium mode.");  
        conditionMode = ConditionMode.MEDIUM;  
    }  
}
```

```

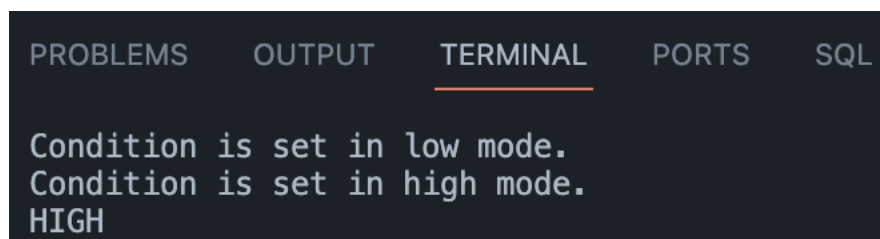
}
}

/**
 * Set the conditioner to high mode if it is not already in high mode.
 */
public void setHighCondition() {
    if (conditionMode == ConditionMode.HIGH) {
        System.out.println("Condition is already in high mode.");
        outputStream.println("Condition is already in high mode.");
    } else {
        System.out.println("Condition is set in high mode.");
        outputStream.println("Condition is set in high mode.");
        conditionMode = ConditionMode.HIGH;
    }
}

/**
 * Close the logger file associated with the conditioner.
 */
public void closeLoggerFile() {
    outputStream.println("Close logger file.");
    outputStream.close();
}
}

```

Виконання програми:



```

PROBLEMS  OUTPUT  TERMINAL  PORTS  SQL
Condition is set in low mode.
Condition is set in high mode.
HIGH

```

Рис. 1. Результат роботи програми.

```
ConditionerLogger1.txt X
ConditionerLogger1.txt
1  Creating a conditioner
2  getFactoryName: #1 Conditioner
3  getConditionColor: WHITE
4  getConditionMode: TURNED_OFF
5  Condition is set in low mode.
6  getConditionMode: LOW
7  Close logger file.
8  |
```

Рис. 2. Результат роботи першого кондиціонера в файлі.

```
ConditionerLogger2.txt X
ConditionerLogger2.txt
1  Creating a conditioner
2  Condition is set in high mode.
3  getConditionMode: HIGH
4  Close logger file.
5  |
```

Рис. 3. Результат роботи другого кондиціонера в файлі.

PACKAGE CLASS TREE INDEX HELP	
PACKAGE: DESCRIPTION RELATED PACKAGES CLASSES AND INTERFACES	
Package KI305.Fedus.Lab2	
package KI305.Fedus.Lab2	
All Classes and Interfaces	
Class	Description
Conditioner	Conditioner class represents a basic air conditioner.
Conditioner.ConditionColor	
ConditionerApp	ConditionerApp class with the entry point for the program.

Рис. 3. Фрагмент згенерованої документації

Контрольні питання:

1. Синтаксис визначення класу.

Відповідь: [public] class НазваКласу

```
{  
  [конструктори]  
  [методи]  
  [поля]  
}
```

2. Синтаксис визначення методу.

Відповідь: [СпецифікаторДоступу] [static] [final] Тип назваМетоду([параметри])
[throws класи]

```
{  
  [Тіло методу]  
  [return [значення]];  
}
```

3. Синтаксис оголошення поля.

Відповідь: [СпецифікаторДоступу] [static] [final] Тип НазваПоля [= ПочатковеЗначення];

4. Як оголосити та ініціалізувати константне поле?

Відповідь: [СпецифікаторДоступу] static final Тип НазваПоля = Значення;

- явно при оголошенні поля класу;

- у статичному блоці ініціалізації.

5. Які є способи ініціалізації полів?

Відповідь:

6. Синтаксис визначення конструктора.

Відповідь: [СпецифікаторДоступу] НазваКласу([параметри])

```
{
```

Тіло конструктора

}

7. Синтаксис оголошення пакету.

Відповідь: `package НазваПакету{.НазваПідпакету};`

8. Як підключити до програми класи, що визначені в зовнішніх пакетах?

Відповідь: вказуючи повне ім'я пакету перед іменем кожного класу або використовуючи оператор `import`

9. В чому суть статичного імпорту пакетів?

Відповідь: Можливість імпортувати окремі статичні методи або поля класу

- `import static`

`НазваПакету{.НазваПідпакету}.НазваКласу.НазваСтатичногоМетодуАбоПоля;`

- `import static НазваПакету{.НазваПідпакету}.*;`

10. Які вимоги ставляться до файлів і каталогів при використанні пакетів?

Відповідь: Для уникнення конфліктів імен не зловживати імпортом пакетів.

Висновок:

Виконавши лабораторну роботу, я ознайомився з процесом розробки класів та пакетів мовою Java.