Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

ПРАКТИЧНА РОБОТА № 2

З дисципліни «Інженерія програмного забезпечення»

на тему «ГРАФІЧНА НОТАЦІЯ UML, ДОКУМЕНТУВАННЯ ПРОЕКТУ»

Виконав:
студент групи ІО-42
Куліков М. М.
Залікова: 4214

Перевірив:
Ст. викладач кафедри ОТ
Васильєва М. Д.

Київ – 2025

Практична робота №2

Тема: «Графічна нотація UML, документування проекту».

Мета: Ознайомитись з видами UML діаграм. Отримати базові навички з використання діаграми класів мови UML. Здобути навички використання засобів автоматизації UML-моделювання. Документування проекту за допомогою JavaDoc.

Виконання роботи:

Завдання 1

1. Побудувати діаграму класів, яка містить три інтерфейси If1, If2, If3 з методами meth1(), meth2(), meth3() та класи що їх реалізують Cl1, Cl2, Cl3, відповідно.
2. Згідно варіанту реалізувати на діаграмі класів відношення генералізації та агрегації.
    a. Для генералізації: (4214%9 = 2) If1 <= If2; If1 <= If3; Cl2 <= Cl3
    b. Для агрегації: (4214%5 = 4) If3 <= Cl2; If2 <= Cl3; Cl3 <= Cl1
3. Розробити інтерфейси і класи згідно діаграми. Реалізація методів має виводити на консоль ім'я класу та назву методу.
4. За допомогою засобів автоматизації UML-моделювання імпортувати сирцевий код та перевірити відповідність побудованої діаграми класів з розробленою. Зберегти діаграму в каталозі документації проекту.
5. Модифікувати сирцевий код, додавши коментарі у форматі JavaDoc. Згенерувати JavaDoc за допомогою Eclipse (або IntelliJIDEA) у каталог документації проекту.
6. Розробити ціль ANT для генерації JavaDoc. Згенерувати JavaDoc за допомогою розробленої цілі ANT.

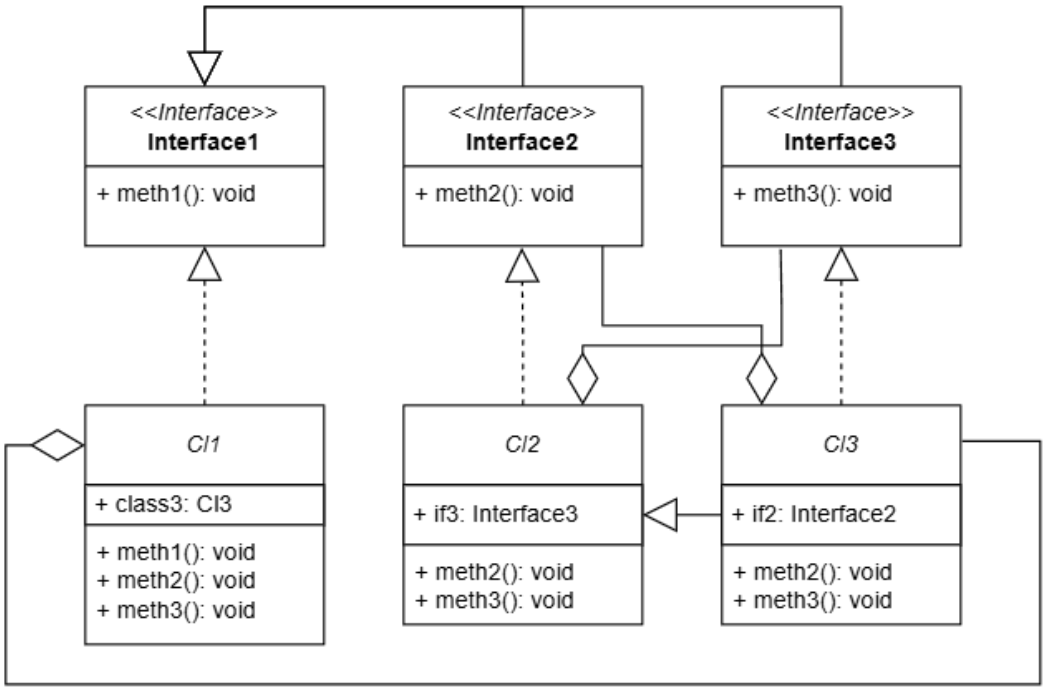

Рисунок 1 – Діаграма класів для розробки коду програми

Вміст файлу main.java:

```
package main.java.com.lab111;

/**
 * Головний клас програми {@code Main}.
 *
 * Клас містить метод {@link #main(String[])}, який демонструє роботу
 * класів {@link Class1}, {@link Class2} та {@link Class3}.
 *
 * Виводить у консоль повідомлення про виклики методів і демонструє
 * агрегацію та реалізацію інтерфейсів у класах проекту.
 *
 * @author Maksym Kulikov (aka @broniev)
 */
public final class Main {
    /**
     * Приватний конструктор для запобігання створенню об’єктів класу.
     */
    private Main() {  super();  }

    /**
     * Демонструє виклики методів класів {@link Class1}, {@link Class2} та {@link Class3}.
     *
     * @param args аргументи командного рядка (не використовуються)
     */
    public static void main(final String[] args) {
        System.out.println("This is MainClass running message!!");
        Class2 cl2 = new Class2();
        Class3 cl3 = new Class3();
        Class1 cl1 = new Class1(cl3);
        System.out.println("> Next messages is called from Class1");
        cl1.meth1();
```

```
        cl1.meth2();
        cl1.meth3();
        System.out.println("> Next messages is called from Class2");
        cl2.meth2();
        cl2.meth3();
        System.out.println("> Next message is called from Class3");
        cl3.meth2();
        cl3.meth3();
    }
}
```

Вміст файлу Interface1.java:

```
package main.java.com.lab111;
/**
 * Інтерфейс {@code Interface1} демонструє приклад базового інтерфейсу.
 * Містить єдиний метод {@link #meth1()}, який має бути реалізований у класах-нащадках.
 */
public interface Interface1 {
    /**
     * Базовий метод інтерфейсу {@code Interface1}.
     * Має бути реалізований у класах, що реалізують цей інтерфейс.
     */
    void meth1();
}
```

Вміст файлу Interface2.java:

```
package main.java.com.lab111;
/**
 * Інтерфейс {@code Interface2} демонструє приклад наслідування інтерфейсів.
 * Інтерфейс {@code Interface2} розширює {@link Interface1},
 * додаючи власний метод {@link #meth2()}.
 */
public interface Interface2 extends Interface1 {
    /**
     * Базовий метод інтерфейсу {@code Interface2}.
     * Має бути реалізований у класах, що реалізують цей інтерфейс.
     */
    void meth2();
}
```

Вміст файлу Interface3.java:

```
package main.java.com.lab111;
/**
 * Інтерфейс {@code Interface3} демонструє приклад наслідування інтерфейсів.
 * Інтерфейс {@code Interface3} розширює {@link Interface1},
 * додаючи власний метод {@link #meth3()}.
 */
public interface Interface3 extends Interface1 {
    /**
     * Базовий метод інтерфейсу {@code Interface3}.
     * Має бути реалізований у класах, що реалізують цей інтерфейс.
     */
    void meth3();
}
```

Вміст файлу Class1.java:

```
package main.java.com.lab111;

/**
 * Клас {@code Class1} демонструє приклад агрегації та реалізації інтерфейсу.
 * Клас реалізує інтерфейс {@link Interface1} та містить атрибут
 * типу {@link Class3}, що демонструє агрегаційний зв’язок.
 * Методи {@code meth2()} делегуються об’єкту {@code class3}.
 */
public class Class1 implements Interface1 {

    /**
     * Атрибут типу {@link Class3}.
     * Реалізує агрегаційний зв’язок: об’єкт {@code Class1}
     * містить посилання на об’єкт {@code Class3}, але не керує його життєвим циклом.
     */
    private Class3 class3;

    /**
     * Конструктор класу {@code Class1}.
     *
     * @param class3 об’єкт типу {@link Class3}, який буде агреговано
     */
    public Class1(Class3 class3) { this.class3 = class3; }

    /**
     * Реалізація методу {@code meth1()} з інтерфейсу {@link Interface1}.
     * Виводить у консоль повідомлення про виклик методу.
     */
    @Override
    public void meth1() {  System.out.println("Class1.meth1()");  }

    /**
     * Делегує виклик методу {@code meth2()} агрегованому об’єкту {@link Class3}.
     * Якщо об’єкт не ініціалізовано, виводить повідомлення про відсутність,
     * інакше виводить у консоль повідомлення про виклик методу.
     */
    public void meth2() {
        if (class3 != null) { class3.meth2(); }
        else { System.out.println("Class3 is'nt initialized");  }  }

    /**
     * Делегує виклик методу {@code meth3()} агрегованому об’єкту {@link Class3}.
     * Якщо об’єкт не ініціалізовано, виводить повідомлення про відсутність,
     * інакше виводить у консоль повідомлення про виклик методу.
     */
    public void meth3() {
         if (class3 != null) {  class3.meth3();  }
        else { System.out.println("Class3 is'nt initialized");     }  } }
```

Вміст файлу Class2.java:

```
package main.java.com.lab111;
/**
 * Клас {@code Class2} демонструє приклад агрегації та реалізації інтерфейсу.
 * Клас реалізує інтерфейс {@link Interface2} та містить атрибут
 * типу {@link Interface3}, що демонструє агрегаційний зв’язок.
 */
```

```
public class Class2 implements Interface2 {

    /**
     * Атрибут типу {@link Interface3}.
     * Реалізує агрегаційний зв'язок: об'єкт {@code Class2}
     * містить посилання на об'єкт, що реалізує {@code Interface3},
     * але не керує його життєвим циклом.
     */
    private Interface3 if3;

    /**
     * Реалізація методу {@code meth1()} з інтерфейсу {@link Interface2}.
     * Виводить у консоль повідомлення про виклик методу.
     */
    @Override
    public void meth1() { System.out.println("Class2.meth1()"); }

    /**
     * Реалізація методу {@code meth2()} з інтерфейсу {@link Interface2}.
     * Виводить у консоль повідомлення про виклик методу.
     */
    @Override
    public void meth2() { System.out.println("Class2.meth2()"); }

    /**
     * Метод класу {@code Class2}.
     * Виводить у консоль повідомлення про виклик методу.
     */
    public void meth3() {  System.out.println("Class2.meth3()");  }
}
```

Вміст файлу Class3.java:

```
package main.java.com.lab111;

/**
 * Клас {@code Class3} демонструє приклад генералізації та агрегації.
 * Генералізація реалізується через наслідування від класу {@link Class2},
 * а агрегація — через наявність атрибута типу {@link Interface2}.
 * Клас також реалізує інтерфейс {@link Interface2}.
 */
public class Class3 extends Class2 implements Interface3 {

    /**
     * Атрибут типу {@link Interface2}.
     * Реалізує агрегаційний зв'язок: об'єкт {@code Class3}
     * містить посилання на об'єкт, що реалізує {@code Interface2}, але не керує його життєвим
циклом.
     */
    private Interface2 interface2;

    /**
     * Метод класу {@code Class3}.
     * Виводить у консоль повідомлення про виклик методу.
     */
    public void meth3() {  System.out.println("Class3.meth3()");    }

    /**
     * Реалізація методу {@code meth2()} з інтерфейсу {@link Interface2}
```

```
     * Перевизначає метод і виводить повідомлення у консоль.
     */
    @Override
    public void meth2() {  System.out.println("Class3.meth2()");    }}
```

Відредагований таргет «generate_javadoc» в build.xml:

```
<!-- ================== generate doc by javadoc Target ======================== -->
      <target name="generate_javadoc"
         description="Zip all project tree from basedir">
      <javadoc sourcepath="${sourceDir}"
            destdir="doc" author="yes"
            version="yes" access="private"
            encoding="UTF-8" docencoding="UTF-8" charset="UTF-8">
      </javadoc>
    </target>
```

Результат виконання програми:

```
PS D:\Documents\Education\KPI\3 Term\IPZ\Projects\PR2\L2_Code> cd out
PS D:\Documents\Education\KPI\3 Term\IPZ\Projects\PR2\L2_Code\out> java -jar .\PR2.jar
This is MainClass running message!!
> Next messages is called from Class1
Class1.meth1()
Class3.meth2()
Class3.meth3()
> Next messages is called from Class2
Class2.meth2()
Class2.meth3()
> Next message is called from Class3
Class3.meth2()
Class3.meth3()
```

Рисунок 2 – результат виконання програми

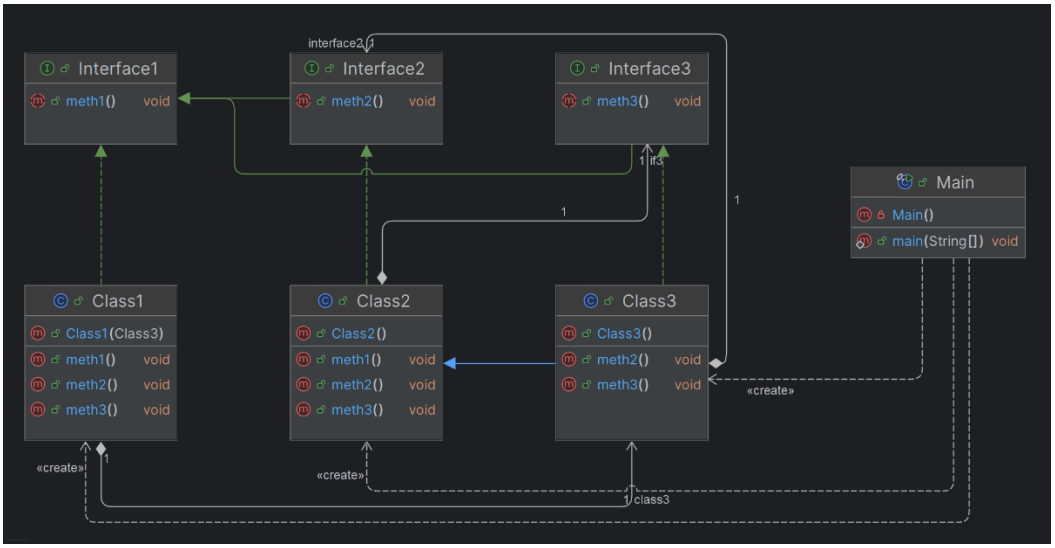

Рисунок 3 – Побудована в IntelliJ IDEA UML діаграма

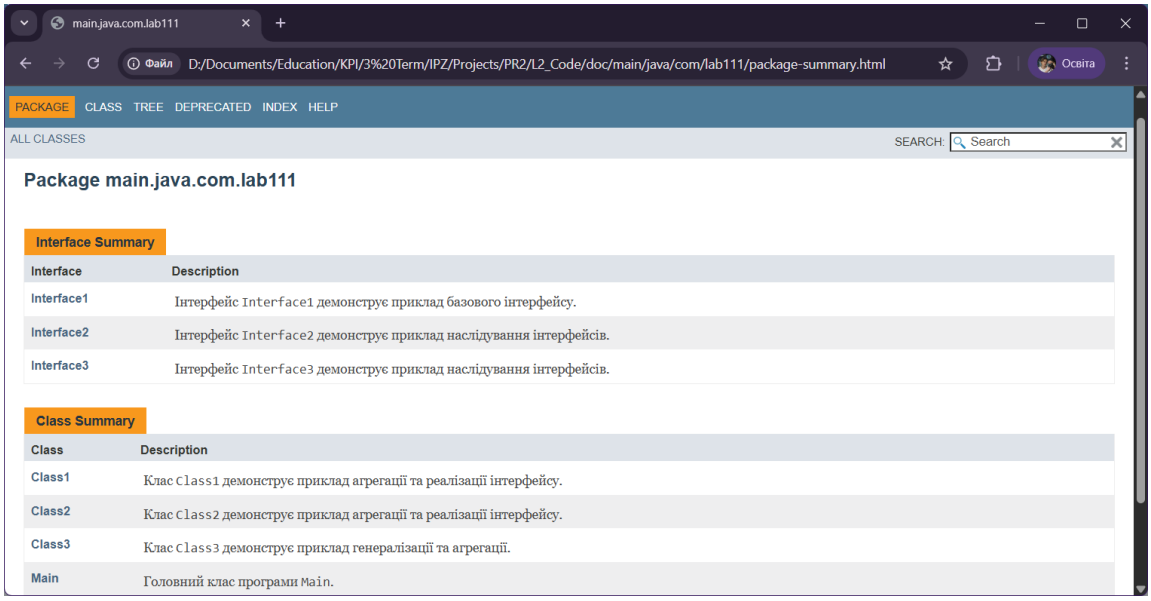

Рисунок 4 – Створена документація у форматі веб сторінки

Висновки:

Отже, під час виконання практичної роботи було освоєно базові навички роботи з UML діаграмами, при чому як в ручну, так і автоматично на основі написаного коду. Порівнюючи вихідну UML діаграму класів з тією, що ми створили на початку (до створення коду), бачимо що вони практично ідентичні, а значить усе зроблено правильно. Єдиним нюансом є те, що виникла потреба переініціалізувати meth1() в класі Class2. Загалом проблем під час виконання практичної роботи не виникло.