

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з дисципліни
«Сучасні методи програмування»
для студентів спеціальності 123 Комп'ютерна інженерія
усіх форм навчання.
Програмування на мові Java

Методичні вказівки до виконання лабораторних робіт з дисципліни «Сучасні методи програмування» для студентів спеціальності 123 Комп'ютерна інженерія усіх форм навчання. Програмування на мові Java / Укл.: Р. К. Кудерметов, О. В. Польська, Н. А. Куликовська — Запоріжжя: НУ «Запорізька політехніка», 2024 — 56 с.

Укладачі: Р. К. Кудерметов, доцент, к.т.н.
О. В. Польська, ст. викладач
Н. А. Куликовська, ст. викладач

Рецензент: М. Ю. Тягунова, доцент, к.т.н.

Відповідальний
за випуск: О. В. Польська, ст. викладач

Затверджено:
на засіданні кафедри
«Комп'ютерні системи та мережі»
Протокол № 2 від 26 вересня 2024 р.

Рекомендовано до видання:
НМК факультету КНТ
Протокол № 2 від 28 жовтня 2024 р.

Зміст

1	Основи мови Java	4
1.1	Класи та об'єкти в Java	4
1.1.1	Програма: класи <code>Matrix</code> , <code>Calculation</code> , <code>Test</code> . .	4
1.2	Спадкування в Java	8
1.2.1	Програма <code>Editor</code>	10
1.3	Зміст звіту	19
1.4	Контрольні питання	19
2	Основи роботи з XML документами	21
2.1	Теоретичні відомості	21
2.2	Практичне завдання	30
2.2.1	Проект з DTD	30
2.2.2	Проект з XML Schema	33
2.3	Зміст звіту	38
2.4	Контрольні питання	38
3	Практична робота в Java	40
3.1	Утиліта <code>ant</code>	40
3.1.1	Програма “Паліндром”	40
3.2	Файлове введення-виведення у Java	45
3.2.1	Програма роботи з файлами	45
3.3	Документування коду	48
3.3.1	Програма для обчислення числа Ейлера	49
3.4	Зміст звіту	54
3.5	Контрольні питання	55
	Рекомендована література	56

ЛАБОРАТОРНА РОБОТА 1

Основи мови Java

Мета роботи — отримати початкові знання про мову Java, її синтаксис, організацію застосунків та реалізацію в ній фундаментальних парадигм об'єктно-орієнтованого програмування — інкапсуляції, спадкування та поліморфізму.

Надбані знання дозволять надалі, за необхідності, опанувати інші потужні можливості мови Java та її численних бібліотек та фреймворків [1, 2].

1.1 Класи та об'єкти в Java

1.1.1 Програма: класи *Matrix*, *Calculation*, *Test*

У першій програмі даної роботи ми вивчимо основний синтаксис мови Java, приділивши увагу процесу створення виконуваного Java-застосунку, що можна здійснити безпосередньо з командного рядка. Знання цього процесу є необхідним для вивчення послідовності реалізації програм на Java і допоможе правильно розуміти методологію створення застосунків в інтегрованих середовищах (Eclipse, IntelliJ IDEA, NetBeans) і з допомогою засобів автоматизації розробки (ant, maven).

Програма складається з трьох класів: *Matrix* (лістинг 1.1), *Calculation* (лістинг 1.2) та *Test* (лістинг 1.3). У класі *Matrix* визначено конструктор, який створює квадратну матрицю з розмірністю, що задається параметром конструктора. Клас має два переважаних методи: `fillMatrix()` і `fillMatrix(int)`. Перший метод заповнює матрицю випадковими цілими числами, другий - робить матрицю діагональною, заповнюючи елементи головної діагоналі значенням параметра методу, а інші елементи — нулями. У класі також визначено метод виведення матриці на друк `printMatrix()`, який використовує при цьому статичний метод `toString()` класу *Arrays*.

Лістинг 1.1 Клас Matrix, файл Matrix.java

```
1  import java.util.Arrays;
2
3  public class Matrix {
4      private int n;
5      public int [][] matrix;
6      public Matrix(int n){
7          this.n = n;
8          matrix = new int[n][n];
9      }
10     public void fillMatrix(){
11         for(int i = 0; i < n; i++)
12             for(int j = 0; j < n; j++)
13                 matrix[i][j] =
14                     (int)(Math.random()*10);
15     }
16     public void fillMatrix(int k){
17         for(int i = 0; i < n; i++)
18             for(int j = 0; j < n; j++){
19                 if(i == j)
20                     matrix[i][j] = k;
21                 else
22                     matrix[i][j] = 0;
23             }
24     }
25     public void printMatrix(){
26         for(int i = 0; i < n; i++){
27             System.out.println(Arrays.toString(matrix[i]));
28         }
29         System.out.println();
30     }
```

Клас програми **Calculation** містить метод **multiMatrix()**, призначений для множення матриць. Метод має тип **Matrix** (повертає результат множення матриць), в якості вхідних параметрів – дві матриці (посилання на об'єкти типу **Matrix**), які треба перемножити, та їх розмірність.

Лістинг 1.2 Клас Calculation, файл Calculation.java

```
1 public class Calculation {
2     public Matrix multiMatrix(Matrix a, Matrix b,
3         int n){
4         Matrix c = new Matrix(n);
5         for(int i = 0; i < n; i++)
6             for(int j = 0; j < n; j++)
7                 for(int k = 0; k < n; k++)
8                     c.matrix[i][j] +=
9                         a.matrix[i][k]*b.matrix[k][j];
10    }
11 }
```

Клас `Test` містить головний метод програми `main()` або точку входу в програму. Такі класи іноді називають головними. У методі `main()` реалізовано зчитування параметрів командного рядка, серед яких розмір матриць перетворюється на ціле число за допомогою методу `parseInt()` класу-оболонки `Integer`. У класі демонструється створення об'єктів класів та використання їх методів.

Лістинг 1.3 Клас Test, файл Test.java

```
1 public class Test {
2     public static void main(String[] args) {
3         System.out.println("My name is " + args[0]);
4         System.out.println("My group is " +
5             args[1]);
6         System.out.println("Number of words in
7             comandline is: " + args.length);
8         System.out.println("size of matrix = " +
9             args[2]);
10        int n=Integer.parseInt(args[2]);
11        Matrix m1 = new Matrix(n);
12        Matrix m2 = new Matrix(n);
13        Matrix m3 = new Matrix(n);
14        Calculation c1 = new Calculation();
15        m1.fillMatrix();
16        m1.printMatrix();
17    }
```

```

14         m2.fillMatrix(1);
15         m2.printMatrix();
16         m3 = c1.multiMatrix(m1,m2,n);
17         m3.printMatrix();
18     }
19 }

```

Завдання до програми

1. Вивчіть лістинги 1.1, 1.2 та 1.3;

2. Реалізуйте програму у **Командному рядку** (cmd.exe).

Для цього виконайте наступне:

2.1 У робочому каталозі створіть окрему папку для проекту, наприклад Lab41;

2.2 У будь-якому текстовому редакторі (Notepad, Notepad++ тощо, але не в Microsoft Word) створіть три файли **Matrix.java**, **Calculation.java** та **Test.java**, вміст яких відповідає лістингам 1.1, 1.2 та 1.3;

2.3 Зайдіть в папку Lab41 у **Командному рядку** і відкомпілюйте окремо кожен із цих файлів або тільки файл **Test.java** (у цьому випадку інші файли також відкомпілюються):

```
javac Test.java
```

2.4 Після компіляції виконайте команду

```
jar cvfe test.jar Test *.class
```

В цій команді test.jar – ім'я файлу, що виконується, Test — ім'я головного класу. В результаті у каталозі Lab41 буде створено виконуваний файл, вміст якого можна переглянути за допомогою будь-яких архіваторів (**ZIP**, **RAR**, **JAR**). У цьому файлі будуть архівовані файли **Calculation.class**, **Matrix.class** та **Test.class**, а також папка **META-INF** з файлом **MANIFEST.MF**, в якому вказано головний клас та наведено іншу інформацію;

2.5 Перевірте працездатність виконуваного файлу за допомогою команди:

```
java -jar test.jar Petrenko KHT535 4
```

3. Доопрацюйте програму згідно з варіантами.

Варіант 1. У класі **Calculation** створіть методи для складання та

транспонування матриць. У головному класі продемонструйте працездатність цих методів. Для цього створіть матрицю `m4`, занесіть у неї результат обчислень суми матриць `m3` і `m1`, потім транспонуйте матрицю `m4` і виведіть на екран результати обчислень.

Варіант 2. Допрацюйте програму таким чином, щоб вона працювала з прямокутними матрицями, причому при множенні матриць необхідно забезпечити перевірку їхньої узгодженості (кількість стовпців першої матриці має дорівнювати числу рядків другої матриці).

4. Розробіть власний приклад програми згідно з варіантами.

Варіант 1. У програмі створіть масив із 25 елементів типу `double` і заповніть його випадковими числами в діапазоні від 0 до 1. Використовуючи оператор `if`, розподіліть елементи масиву на елементи, які більші за його другий елемент і які менше другого елементу, помістивши їх у відповідні два додаткові масиви. Виведіть усі масиви на друк у вигляді трьох стовпців. В якості оператора циклу використовуйте `while`.

Варіант 2. У програмі створіть масив із 25 елементів типу `int` і заповніть його випадковими числами в діапазоні від 0 до 100. Розподіліть елементи масиву на парні та непарні, помістивши їх у відповідні два додаткові масиви. Створіть ще один додатковий масив, до якого занесіть елементи з парними індексами. Виведіть усі масиви на друк у вигляді чотирьох стовпців.

Примітка. В якості генератора випадкових чисел можна використовувати статичний метод `Math.random()`, який генерує випадкові числа типу `double` в діапазоні від 0 до 1. Для отримання цілих випадкових чисел цим методом слід застосовувати перетворення типу.

5. Вивчіть опції програми `jar` (команда `jar -help`);

6. Реалізуйте програму у інтегрованому середовищі розробки, наприклад IDE Eclipse.

1.2 Спадкування в Java

Цей розділ присвячений ідеології та техніці спадкування, що реалізоване в мові Java, та використанню його для організації ієрархії класів та поліморфізму в Java-застосунках.

З поняттям спадкування ви вже знайомі після виконання лабораторних робіт, присвячених об'єктно-орієнтованій мові C++. Незважаючи на те, що спадкування у багатьох об'єктно-орієнтованих мовах має багато спільного, проте реалізація спадкування в мові Java має свої суттєві і не тільки синтаксичні особливості, які слід ретельно вивчити та зрозуміти для оволодіння технікою сучасного програмування мовою Java.

Говорячи про спадкування, програмісти часто використовують для батьківських класів термін *суперклас*, а для похідних — *підклас*. Для вказування спадкування в мові Java використовується ключове слово **extends**, яке, до того ж підкреслює, що підклас «розширює» суперклас новими властивостями та методами. Наприклад:

```
public class Derived extends Base{
    additional fields and methods;
}
```

У мові Java відсутнє пряме множинне спадкування. Тому кожен клас може мати лише один безпосередній суперклас. Для його позначення використовується ключове слово **super**, за допомогою якого в підкласі можна отримати доступ до відкритих та захищених полів та методів суперкласу, а також викликати його конструктор (цей виклик має бути першим у тілі конструктора похідного класу). Наприклад:

```
public class Person{
    private String name;
    public Person(String name){
        this.name = name;
    }
    public String getInfo(){
        return name;
    }
}

public class Student extends Person{
    private int group;
    public Student(String name, int group){
        super(name);
        this.group = group;
    }
}
```

```

    }
    public String getInfo(){
        return super.getInfo() + " " +
            Integer.toString(group);
    }
}

```

1.2.1 Програма Editor

У цій програмі демонструється механізм спадкування мови Java. Моделлю програми є сімейство текстових редакторів. Розширюючи властивості простого текстового редактора, можна створити різні редактори, зокрема редактор до створення офісних документів, редактори, які входять до складу інтегрованих середовищ розробки програм (IDE) тощо. Діаграма класів моделі наведена на рис. 1.1.

Зверніть увагу на розподіл ролей між класами. За реалізацію функцій введення/виведення тексту відповідає клас `TextEditor`. Його підкласом є клас `NotePad`, в якому, додатково до успадкованих від класу `TextEditor` методів введення/виведення тексту, є метод для створення документа (примірника класу `Document`) та збереження його за допомогою статичного методу `saveAs()` класу `FileManager`.

Клас `OfficeEditor` розширює властивості класу `NotePad`, він додає можливість встановлювати шрифт тексту (`font`) та ліве поле документа (`leftMargin`).

Класи `IDECpp` та `IDEJava` також використовують можливості класу `NotePad`, наслідуючи його опосередковано через абстрактний клас `IDE`. Абстрактний клас `IDE` містить єдиний метод `doCompile()`, що є абстрактним. У кожному з класів `IDECpp` та `IDEJava` цей метод перевизначається та імітує функцію компіляції створених у IDE текстів програм. За допомогою цього абстрактного методу організовано динамічний поліморфізм, який продемонстровано у головній функції програми (лістинг 1.12, рядки 17-19).

У лістингах (1.4 - 1.12) представлені класи програми. Головну функцію програми наведено у лістингу 1.12.

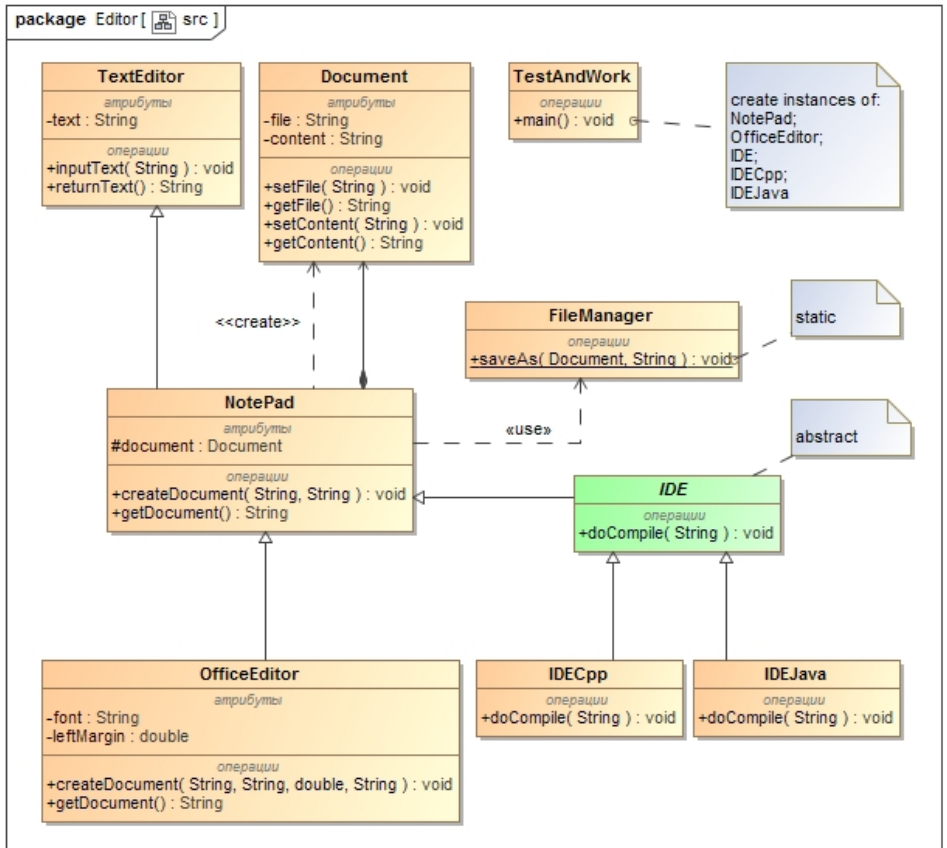


Рисунок 1.1 – UML-діаграма класів програми Editor

ЛІСТИНГ 1.4 Клас Document

```

1 package data;
2
3 public class Document {
4     private String file;
5     private String content;
6     public void setFile(String file){
7         this.file = file;

```

```

8      }
9      public void setContent(String content){
10         this.content = content;
11     }
12     public String getFile(){
13         return file;
14     }
15     public String getContent(){
16         return content;
17     }
18 }

```

Лістинг 1.5 Клас TextEditor

```

1  package texteditor;
2
3  public class TextEditor {
4      private String text;
5      public void inputText(String text){
6          this.text = text;
7      }
8      public String returnText(){
9          return text;
10     }
11 }

```

Лістинг 1.6 Клас FileManager

```

1  package filemanager;
2
3  import java.text.SimpleDateFormat;
4  import java.util.Date;
5  import data.Document;
6
7  public class FileManager{
8      public static void saveAs(Document document,
9          String fileName){
10         SimpleDateFormat simpleDate = new
11             SimpleDateFormat("dd.MM.yy k:m");

```

```

10         String file = fileName + " " +
            simpleDate.format(new Date());
11         document.setFile(file);
12     }
13 }

```

ЛІСТИНГ 1.7 Клас NotePad

```

1  package texteditor;
2
3  import data.Document;
4  import filemanager.FileManager;
5
6  public class NotePad extends TextEditor{
7      protected Document document;
8      public void createDocument(String text, String
          fileName){
9          document = new Document();
10         inputText(text);
11         document.setContent(returnText());
12         FileManager.saveAs(document, fileName);
13     }
14     public String getDocument(){
15         return "Plain text created: " +
            document.getContent() + " " +
            document.getFile();
16     }
17 }

```

ЛІСТИНГ 1.8 Клас OfficeEditor

```

1  package office;
2
3  import texteditor.*;
4
5  public class OfficeEditor extends NotePad {
6      private String font;
7      private double leftMargin;
8

```

```

9      public void createDocument(String text, String
        font, double leftMargin, String fileName){
10      createDocument(text, fileName);
11      this.font = font;
12      this.leftMargin = leftMargin;
13  }
14  public String getDocument(){
15      String properties = font + " " +
        Double.toString(leftMargin) + " ";
16      return "Office document created:
        "+document.getContent()+ " "+
        properties+document.getFile();
17  }
18  }

```

Лістинг 1.9 Абстрактний клас IDE

```

1  package ide;
2
3  import texteditor.NotePad;
4
5      public abstract class IDE extends NotePad {
6          public abstract void doCompile(String s);
7  }

```

Лістинг 1.10 Клас IDECpp

```

1  package ide;
2
3  public class IDECpp extends IDE {
4      public void doCompile(String name){
5          System.out.println("Compile output is " +
            name + ".exe");
6      }
7  }

```

Лістинг 1.11 Клас IDEJava

```

1  package ide;
2

```

```

3 public class IDEJava extends IDE{
4     public void doCompile(String name){
5         System.out.println("Compile output is " + name
6             + ".class");
7     }
8 }

```

Лістинг 1.12 Клас TestAndWork з точкою входу до програми

```

1 package testandwork;
2
3 import office.*;
4 import texteditor.*;
5 import ide.*;
6
7 public class TestAndWork {
8     public static void main(String[] args){
9         NotePad note = new NotePad();
10        note.createDocument("text text text",
11            "myfile.txt");
12        System.out.println(note.getDocument());
13
14        OfficeEditor ofe = new OfficeEditor();
15        ofe.createDocument("doc doc doc", "Times
16            Roman", 2.5, "myfile.doc");
17        System.out.println(ofe.getDocument());
18
19        IDE[] ide = new IDE[2];
20        ide[0] = new IDECpp();
21        ide[1] = new IDEJava();
22
23        String programText[] = {"class A: public B
24            {...};" , "class A extends B {...} "};
25        String program[] = {"lab33", "lab42"};
26        String extension[] = {".cpp", ".java"};
27
28        for(int i = 0; i < 2; i++){
29            ide[i].createDocument(programText[i],
30                program[i] + extension[i]);
31            System.out.println(ide[i].getDocument());
32        }
33    }
34 }

```

```

28         ide[i].doCompile(program[i]);
29     }
30 }
31 }

```

В результаті правильної реалізації програми та її виконання на консоль повинні вивестись (з точністю до дати та часу) повідомлення (лістинг 1.13):

Лістинг 1.13 Результат реалізації програми Editor

```

Plain text created: text text text myfile.txt
04.11.24 13:31
Office document created:
doc doc doc Times Roman 2.5 myfile.doc 04.11.24
13:31
Plain text created:
class A: public B {...}; lab33.cpp 04.11.24 13:31
Compile output is lab33.exe
Plain text created:
class A extends B {...} lab42.java 04.11.24 13:31
Compile output is lab42.class

```

Завдання до програми

1. Вивчіть UML-діаграму класів та код програми, реалізуйте проект програми, наприклад у IDE Eclipse;

2. Створіть виконуваний jar-файл цієї програми. Для цього виконайте наступну послідовність (передбачається, що каталог проекту має назву **Editor**, див. рис. 1.2):

2.1 Оберіть проект (пакет **Editor** у полі **Package Explorer**) і в контекстному меню проекту натисніть **Refresh** (Оновити), щоб у виконуваний файл потрапили актуальні версії класів;

2.2 Знову відкрийте контекстне меню проекту та виберіть команду **Export**. У вікні, що відкриється, знайдіть папку **Java** і в ній оберіть опцію **Runnable JAR file** (див. рис. 1.3);

2.3 У вікні **Runnable JAR file Export**, що відкриється (див. рис. 1.4), зробіть наступне:

- задайте розташування та ім'я виконуваного jar-файлу у полі **Export destination**, наприклад, `Editor\bin\editor.jar`;
- виберіть головний клас (`TestAndWork`) у полі **Launch configuration**;
- забезпечте для перемикача **Library handling** стан **Extract required libraries into generated JAR** (Вилучити необхідні бібліотеки) і натисніть **Finish**;

2.4 Перевірте працездатність виконуваного jar-файлу. Для цього у **Командному рядку** зайдіть у папку, де розміщений цей файл, та виконайте команду:

```
java -jar editor.jar
```

3. *Розробіть власний приклад програми. Для цього додайте до програми клас, що моделює офісний додаток «табличний процесор» (табличні процесори — програми типу Microsoft Excel), який будь-яким чином розширює клас `NotePad`. Повторіть дії з реалізації проекту, створення і перевірки працездатності виконуваного jar-файлу.

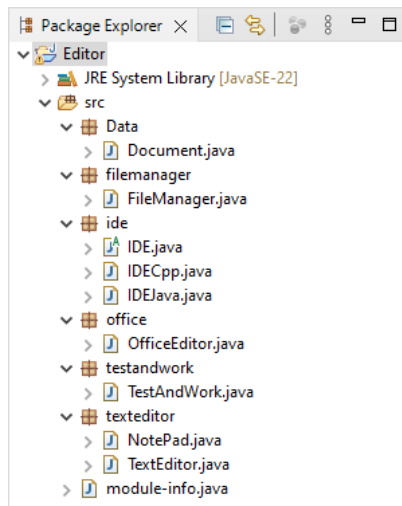
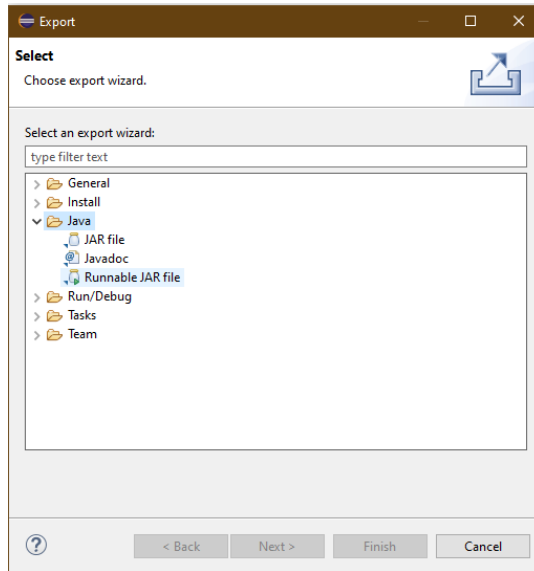
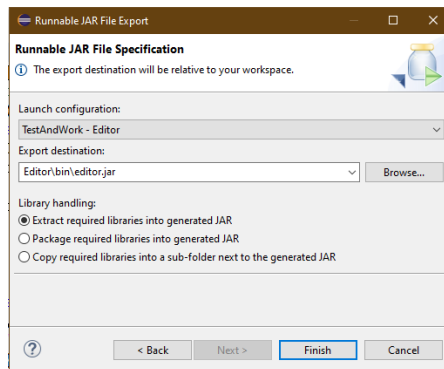


Рисунок 1.2 – Структура проекту в IDE Eclipse

Рисунок 1.3 – Вікно команди **Export**Рисунок 1.4 – Вікно **Runnable JAR file Export**

1.3 Зміст звіту

Звіт повинен містити: мету роботи, лістинги або фрагменти коду, що розроблені для власного прикладу програм, з необхідними поясненнями та відповіді на контрольні питання.

1.4 Контрольні питання

1. Які компоненти можуть бути описані у класі? Навести приклад опису класу із названими компонентами.
2. Навести синтаксис створення екземпляра класу.
3. Як у Java створити «багатовимірний» масив? Навести приклад.
4. Що таке *суперклас* та *підклас* у Java?
5. Чи можна у мові Java створювати перевантажені методи? Якщо так, то навести приклад.
6. Чи можна у підкласі перевизначити метод суперкласу?
7. Що таке статичні поля та методи класу і як вони використовуються?
8. Чи можна перевизначити статичний метод класу? Якщо так, то навести приклад.
9. Чому метод `main()` визначається як статичний?
10. Чи може метод `main()` бути розміщеним у декількох класах програми?
11. Що таке *термінальний клас* та *термінальний метод* класу?
12. Чи можна у підкласі звернутися до методу суперкласу?
13. Що таке *абстрактний клас* у Java?
14. Як визначити *абстрактний метод* класу?
15. Чи можна визначити конструктор у абстрактному класі?
16. Навести та описати кілька методів класу `Object`.
17. Як викликати у підкласі конструктор суперкласу та передати до нього параметри?
18. Що таке *інтерфейс* у Java? Навести приклад інтерфейсу.
19. Як реалізувати методи, оголошені в інтерфейсі? Навести приклад.

20. Навіщо в Java застосовують абстрактні класи та інтерфейси?
21. Чи можна між інтерфейсами встановити відношення спадкування?
22. Як у Java реалізується динамічний поліморфізм?
23. Чи може одночасно реалізовуватись кілька інтерфейсів?
24. Чи може клас одночасно успадковувати суперклас та реалізовувати інтерфейс?
25. Нехай оголошено інтерфейс. Чи можна змінну, що має тип цього інтерфейсу (*інтерфейсне посилання*), використовувати як посилання на об'єкт, що реалізує цей інтерфейс?
26. Що таке *пакети* у Java і для чого вони використовуються? Навести приклад програми із двома пакетами.
27. Чи може клас належати кільком пакетам одночасно?
28. Скільки інструкцій **package** може бути у файлі?
29. Чи можна посилатися на клас у іншому пакеті без використання ключового слова **import**;
30. Заповніть таблицю 1.1: ставте знак плюс, якщо поле доступне, і знак мінус - якщо інакше.

Таблиця 1.1 – Рівні доступу полів (методів) класу

Ідентифікатор до- ступу	private	відсутній	protected	public
Той самий клас	+			+
Підклас того ж пакету				
Не підклас того ж па- кету				
Підклас іншого пакету				
Не підклас іншого па- кету				

ЛАБОРАТОРНА РОБОТА 2

Основи роботи з XML документами

Мета роботи — ознайомитися з основними принципами створення XML документів та схем XML Schema, навчитися використовувати опис DTD та таблиці стилів XSL.

2.1 Теоретичні відомості

XML (*EXtensible Markup Language*, розширювана мова розмітки) – стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, розроблений та запропонований консорціумом World Wide Web Consortium (W3C) [3,4]. Мова XML – мова з простим формальним синтаксисом, зручна для створення та обробки документів як програмами, так і людиною, з акцентом на використання в Інтернеті. Вона є спрощеною версією мови **SGML** (*Standard Generalized Markup Language*, стандартна узагальнена мова розмітки), яку використовували для опису документів зі складною структурою з високими вимогами до документації.

XML документи мають як фізичну, так і логічну структуру. Фізично документ складається з одиниць, які називають сутностями (*Entity*), мають вміст і ідентифікуються назвою сутності. Кожен XML-документ має одну головну сутність, яка називається сутністю документа і може містити весь документ. Інші можливі сутності вказуються за допомогою посилання на сутності.

Кожен XML документ має ієрархічну логічну структуру та складається з одного або більше *елементів*, які обмежені *відкриваючими* (*start-tags*) та *закриваючими тегоми* (*end-tags*). Тип елемента визначений його ім'ям або ідентифікатором і може мати набір специфікацій атрибутів. Специфікація атрибута має назву (*name*) та значення (*value*). Приклад XML документу див. лістинг 2.1. Деякі пояснення:

- рядок 1 містить вказівки (*Processing Instruction*) щодо обробки документу. Обробник `<?xml?>` задає параметри XML-документу.

Атрибут **version** вказує версію XML, на даний момент вона повинна бути “1.0”, **encoding** вказує формат кодування, набір символів, що використовується в документі. Атрибут **standalone**, який може приймати значення “yes” або “no”, визначає, чи може даний документ бути оброблений без використання будь-яких інших файлів;

- рядки 2-18 містять один кореневий елемент і піделементи, структура яких складається з коректної пари відкриваючого та закриваючого тегів `<назва-тегу> ... </назва-тегу>`;

- рядки 3, 8, 13 містять атрибути, тобто пари ключ-значення, які знаходяться у середині тега

`<назва-тегу назва-атрибута = “значення атрибута”/>`.

Лістинг 2.1 Приклад XML документу

```

1  <?xml version="1.0" encoding="UTF-8"
    standalone="yes"?>
2  <garage>
3    <car serial_no="1">
4      <model>BMW X5</model>
5      <year>2023</year>
6      <color>Black</color>
7    </car>
8    <car serial_no="2">
9      <model>Lexus RX</model>
10     <year>2020</year>
11     <color>Grey</color>
12   </car>
13   <car serial_no="3">
14     <model>Audi Q7</model>
15     <year>2022</year>
16     <color>White</color>
17   </car>
18 </garage>

```

Як бачимо, наведений вище XML документ є досить описовим і нічого не виконує, це просто інформація загорнута в теги.

Мова XML використовується для зберігання та пересилання структурованих даних і є популярною як метод перетворення інформації у ієрархічну форму для зручності зберігання. Мова XML

не має попередньо визначених тегів. Мова не залежить від операційної системи і середовища обробки. Документи XML повинні бути коректними, правильно сформованими, валідними (*valid*). XML документ з лістингу 2.1 правильно сформований, але не є валідним.

Коректний документ XML є валідним, якщо він має пов'язану декларацію типу документу та якщо документ відповідає обмеженням, вираженим у ньому. Існує два різних визначення типу документу, які використовують з XML:

- *Document Type Definition (DTD)*;
- *XML Schema*.

Опис **DTD** – визначення типу документу, який роз'яснює зв'язок тегів між собою та правила їх використання. DTD описує схему документу для певної мови розмітки через набір оголошень (об'єктів-параметрів, елементів та атрибутів), що описують його клас (чи тип) з точки зору синтаксичних обмежень цього документу. Фактично DTD це набір визначень, кожне з яких є елементом.

Оголошення елементів утворює перелік дозволених назв елементів в документі, а також зазначає інформацію щодо тегів (чи є вони обов'язковими) та моделі вмісту для кожного елементу. Приклад визначення елементу **human**, що повинен обов'язково включати елементи **height**, **weight**, **bday** в зазначеному порядку:

```
<!ELEMENT human (height, weight, bday)>
```

Приклад визначення порожнього елементу:

```
<!ELEMENT box EMPTY>
```

Приклад визначення елементу з будь-яким вмістом:

```
<!ELEMENT box ALL>
```

Приклад визначення символічних даних, але не інших елементів:

```
<!ELEMENT str (#PCDATA)>
```

Для елемента можна вказати *кількість входжень* до нього інших елементів за допомогою знаків **?**, **+**, ***** :

- знак **?** говорить про необов'язковість елементу;
- знак **+**, що елемент *повинен* входити хоча б один раз;
- знак *****, що елемент *може* може з'явитись будь-яку кількість разів, у тому числі і нуль.

Приклад використання знаків обмежень кількості входжень :
`<!ELEMENT human (name, second-name?, education+, job*)>`

Оголошення атрибутів – це дозволений набір атрибутів для кожного визначеного елемента, який надає інформацію щодо типу значень атрибутів (чи одразу вказаний список можливих значень) та інформацію щодо потреби встановлювати значення атрибутів за замовчанням.

Приклад визначення двох атрибутів для елемента `computer`, обов’язкового атрибута `model` і необов’язкового `display`:

```
<!ELEMENT computer (#PCDATA)>
```

```
<!ATTLIST computer model CDATA #REQUIRED display CDATA
#IMPLIED>
```

Приклад визначення атрибута `model` для елемента `smartphone` зі списком можливих значень та значенням за замовчуванням `Samsung Galaxy S24`:

```
<!ELEMENT smartphone (#PCDATA)>
```

```
<!ATTLIST smartphone model (Apple_iPhone_16_Pro|
Google_Pixel_9_Pro| Samsung_Galaxy_S24| Sony_Xperia_1_VI)
"Samsung Galaxy S24">
```

DTD до документу XML можна підключити наступним чином:

```
<!DOCTYPE rootElement SYSTEM "example.dtd">
```

де `rootElement` – ім’я кореневого елемента.

Або можна визначити конструкції DTD всередині документу XML (див. лістинг 2.2).

Лістинг 2.2 Визначення DTD в документі XML

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE processorList[
3     <!ELEMENT processorList(processor)+>
4     <!ELEMENT processor
5         (brand,series,model,total_cores,total_threads)>
6     <!ELEMENT brand (#PCDATA)>
7     <!ELEMENT series (#PCDATA)>
8     <!ELEMENT model (#PCDATA)>
9     <!ELEMENT total_cores (#PCDATA)>
```

```

9      <!--ELEMENT total_threads (#PCDATA)-->
10 ]>
11 <processorList>
12   <processor>
13     <brand>Intel</brand>
14     <series>Core i7</series>
15     <model>14700</model>
16     <total_cores>20</total_cores>
17     <total_threads>28</total_threads>
18   </processor>
19 </processorList>

```

XML schema – альтернативний спосіб опису типу документа. У схемі XML визначаються елементи і їх атрибути. Крім того, у схемі XML задаються правила породження елементів (порядок вкладеності елементів, послідовність їх появи) та типізація елементів і атрибутів з можливістю встановлення обмежень на значення елементів і їх атрибутів. Елементи мають обов’язкові атрибути **name** і **type**, що визначають ім’я елемента і його тип. Наприклад:

```
<xs:element name="computer" type="xs:string">
```

У схемах XML підтримуються усі найпоширеніші в мовах програмування типи даних, а саме:

```

xs:string – символний тип;
xs:decimal – десяткове число;
xs:boolean – true/false;
xs:float – число с плаваючою точкою;
xs:date (2020-03-26) – дата;
xs:time (15:30:01) – час.

```

Крім того, кожен елемент може мати атрибути **minOccurs** (мінімальне число входжень елемента) та **maxOccurs** (максимальне число входжень елементів). За замовчуванням ці атрибути мають значення “1”, тобто елемент повинен входити тільки один раз.

Елементи бувають простими і складними. Прості елементи не можуть мати атрибутів і вмісту змішаного типу, а складні – можуть. Вміст змішаного типу – це можливість використовувати усередині елемента дочірні елементи і текст.

Прості елементи визначаються за допомогою вкладеного елемента `<xs:simpleType>`, складні за допомогою `<xs:complexType>`.

Визначення простого елемента символьного типу в XML:

```
<xs:element name="message" type="xs:string"/>
<message>Message text</message>
```

Обмеження на значення елемента можна встановлювати за допомогою конструкції `<xs:restriction>` (див. лістинг 2.3).

Лістинг 2.3 Приклад обмеження на значення елемента

```
1 <xs:element name="age">
2   <xs:simpleType>
3     <xs:restriction base="xs:integer">
4       <!--restriction:-->
5       <xs:minInclusive value="25"/><!--minimum
6         value-->
7       <xs:maxInclusive value="41"/><!--maximum
8         value-->
9     </xs:restriction>
10  </xs:simpleType>
11 </xs:element>
```

Якщо потрібно задати множину можливих значень елемента, слід користуватися `<xs:restriction>` та `<xs:enumeration>` (див. лістинг 2.4).

Лістинг 2.4 Визначення множини можливих значень елемента

```
1 <xs:element name="Processor">
2   <xs:simpleType>
3     <xs:restriction base="xs:string">
4       <xs:enumeration value="INTEL CORE I7"/>
5       <xs:enumeration value="AMD Ryzen 9"/>
6       <xs:enumeration value="INTEL CORE I9"/>
7     </xs:restriction>
8   </xs:simpleType>
9 </xs:element>
```

На відміну від простих елементів складні елементи можуть мати атрибути і вміст змішаного типу. Послідовність появи дочірніх

елементів визначається за допомогою конструкції `<xs:sequence>`. Приклад визначення складного елемента **Computer** зі встановленою послідовністю появи елементів (спочатку **HDD**, потім **Motherboard**) показано у лістингу 2.5. Рядок 6 задає для елемента **Computer** символічний атрибут **model**.

Лістинг 2.5 Приклад визначення складного елемента **Computer**

```

1 <xs:complexType name="Computer">
2   <xs:sequence>
3     <xs:element name="HDD" type="xs:string"/>
4     <xs:element name="Motherboard"
        type="xs:string"/>
5   </xs:sequence>
6   <xs:attribute name="model" type="xs:string"/>
7 </xs:complexType>

```

Визначення атрибуту `<xs:attribute/>` також може мати параметр **use**, який може приймати значення “**required**” (обов’язковий атрибут), “**optional**” (необов’язковий), “**prohibited**” (заборонений). Параметр **default** задає значення атрибуту за замовчуванням.

Бувають випадки, коли треба визначити атрибут для простого елемента. Наприклад, у рядок 6, що визначає ціну на товар (див. лістинг 2.6), потрібно додати атрибут, що буде вказувати на валюту. Щоб додати оголошення атрибуту потрібно визначити комплексний тип. Щоб вміст елемента був простим числом типу **decimal** можна отримати новий комплексний тип з простого типу. Для опису нового анонімного типу використано елемент **complexType**. Щоб вказати, що новий тип містить лише символічні дані і не містить піделементів, використано елемент **simpleContent**. Розширення типу **decimal** полягає в додаванні атрибуту **currency** (шляхом використання стандартного оголошення).

Лістинг 2.6 Приклад отримання складного типу з простого

```

1 <xs:element name="price" >
2   <xs:complexType>
3     <xs:simpleContent>

```

```

4      <xs:extension base="xs:decimal">
5      <xs:attribute name="currency"
           type="xs:string"/>
6      </xs:extension>
7  </xs:simpleContent>
8  </xs:complexType>
9 </xs:element>

```

Екземпляр XML Schema називається визначенням XML схеми (*XML Schema Definition, XSD*) і, зазвичай, має розширення в імені файлу “.xsd”.

Підключити XML Schema до XML можна так:

```

<rootElement
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xsi:noNamespaceSchemaLocation='example.xsd'>
  ...
</rootElement>

```

Таблиці стилів *XSL (EXtensible Stylesheet Language)* описують, як документи XML повинні перетворюватися в інші формати, наприклад HTML. Таблиці стилів XML не просто перетворювачі форматів, вони також надають механізм для маніпулювання даними. Дані можна сортувати, робити по ним пошук, видаляти або додавати прямо з браузера.

XSL спроможна також здійснювати умовну трансформацію виведення в залежності від значень різноманітних елементів або атрибутів. Більш того, вона дозволяє запитувати дані з використанням множини різноманітних операторів шаблонів, символів підстановки, фільтрів, булевих операторів і виражень множини. Таблиця стилів використовує як власні теги, так і запозичені з HTML.

Розглянемо приклади використання тегів таблиці стилів (див. лістинги 2.7-2.8). Тег `<xsl:template>` визначає шаблон, створений для багаторазового використання та форматування виведення вузлів визначеного типу у визначеному контексті. Його атрибут `match` вказує на елемент, вміст якого буде сформований основним шаблоном документу, тобто кореневий елемент. Тег `<xsl:for-each>` вказує, що шаблон буде використовуватися для кожного елементу, вказаного у атрибуті `select`.

 Лістинг 2.7 Приклад використання тегів

```

1 <xsl:template match="articles">
2   <xsl:for-each select="article">
3     <div class="news_body" id="id_{@id}">
4       <xsl:apply-templates select=
5         "date|title|image|description|links"/>
6     </div>
7   </hr/>
8 </xsl:for-each>
9 </xsl:template>

```

Завдяки тегу `<xsl:apply-templates>` процесор шукає шаблон для кожного з вузлів, які вказані або в DTD, або в XSD документі. Після того, як було оголошено основний шаблон, за допомогою тегу `<xsl:template>` визначаються шаблони кожного елемента. Тег `<xsl:value-of>` вставляє значення обраного вузла у вигляді тексту.

 Лістинг 2.8 Приклад використання тегу `<xsl:apply-templates>`

```

1 <xsl:template match="article">
2   <xsl:apply-templates select="."/>
3 </xsl:template>
4 <xsl:template match="date">
5   <p><xsl:value-of select="."/></p>
6 </xsl:template>
7 <xsl:template match="title">
8   <h1><xsl:value-of select="."/></h1>
9 </xsl:template>

```

Декілька прикладів визначення шаблону для рисунків у XSL-документі (див. лістинги 2.9-2.10)

 Лістинг 2.9 Визначення шаблону для рисунків (приклад 1)

```

1 <xsl:template match="image">
2   <center>
3     
4   </center><br/>
5 </xsl:template>

```

Лістинг 2.10 Визначення шаблону для рисунків (приклад 2)

```
1 <xsl:template match="links">
2   <div class="links">
3     <ul>
4       <xsl:for-each select="link">
5         <li><xsl:apply-templates select="."/></li>
6       </xsl:for-each>
7     </ul>
8   </div>
9 </xsl:template>
10 <xsl:template match="link">
11   <a href="{@href}"><xsl:value-of
12     select="."/></a>
12 </xsl:template>
```

2.2 Практичне завдання

2.2.1 Проект з DTD

Створити DTD-документ, який описує схему для створення документу резюме, що містить наступну інформацію:

- ім'я (обов'язково);
- прізвище (обов'язково);
- по-батькові (не обов'язково);
- день народження (обов'язково);
- записи про освіту (мінімум один);
- записи про роботу (будь-яка кількість).

Створити XML-документ з прикладом резюме, який використовує правила розмітки DTD.

Для цього у середовищі розробки створити новий Java-проект з ім'ям *task1*. До проекту додати необхідні файли (див. лістинги 2.11-2.12). Перевірити відповідність XML документа його типу, який описаний у файлі `dtd_resume.dtd`. Отримати повідомлення про успішну перевірку (валідацію) XML файлу. Приклад виконання завдання у середовищі NetBeans (рис. 2.1) та Liquid Studio (рис. 2.2) [5, 6].

ЛІСТИНГ 2.11 Файл resume.xml

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <!DOCTYPE resume "dtd_resume.dtd">
3  <resume>
4      <lastName>Petryk</lastName>
5      <firstName>Pavlo</firstName>
6      <initial>Petrovych</initial>
7      <bday>October 26, 2000</bday>
8      <edu level="school">
9          Zaporizhzhia Lyceum No.71
10     </edu>
11     <edu level="college">
12         Zaporizhzhia Vocational College of
13         Computer Technologies,
14         Computer engineering
15     </edu>
16     <edu level="university">
17         Zaporizhia Polytechnic National
18         University, Computer Engineering
19     </edu>
20     <edu level="university">
21         National Aviation University,
22         Cybersecurity and Information
23         Protection
24     </edu>
25     <job>
26         <company>Freshcode</company>
27         <position>Java Developer</position>
28         <start>November 25, 2021</start>
29         <end>June 14, 2023</end>
30     </job>
31     <job>
32         <company>EPAM Ukraine</company>
33         <position>Java Developer</position>
34         <start>June 24, 2023</start>
35         <end>January 15, 2024</end>
36     </job>
37 </resume>

```

Лістинг 2.12 Файл dtd_resume.dtd

```

1  <!ELEMENT resume
   (lastName,firstName,initial?,bday,edu+,job*)>
2  <!ELEMENT lastName (#PCDATA)>
3  <!ELEMENT firstName (#PCDATA)>
4  <!ELEMENT initial (#PCDATA)>
5  <!ELEMENT bday (#PCDATA)>
6  <!ELEMENT edu (#PCDATA)>
7  <!ATTLIST edu level
   (school|college|university|academy)"university">
8  <!ELEMENT job (company,position,start,end)>
9  <!ELEMENT company (#PCDATA)>
10 <!ELEMENT position (#PCDATA)>
11 <!ELEMENT start (#PCDATA)>
12 <!ELEMENT end (#PCDATA)>

```

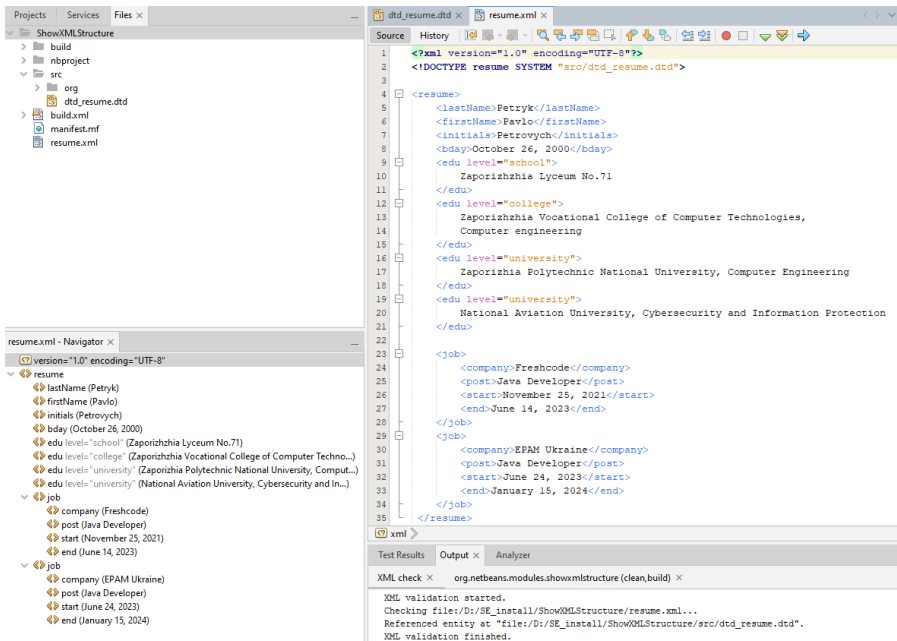


Рисунок 2.1 – Приклад виконання завдання в IDE NetBeans

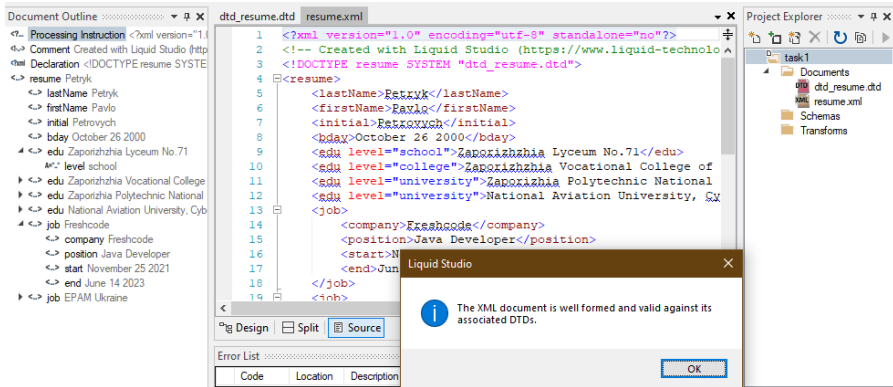


Рисунок 2.2 – Виконання завдання у середовищі Liquid Studio

Завдання до проекту

Варіант 1. Доповнити XML документ можливістю додавати записи про контакти елементом `<contact> </contact>`, тип контакту визначити атрибутом `type`, який може приймати значення `tel`, `address`, `email` або `messenger`. Значення атрибуту `type` за замовчуванням повинно бути `tel`. Зробити обов'язковим для введення мінімум одного запису про контакт.

Варіант 2. Доповнити XML документ можливістю додавати записи про контакти елементом `<contact> </contact>`, який може містити в собі елементи `<tel> </tel>`, `<address> </address>`, `<email> </email>`, `<messenger> </messenger>`. Зробити так, щоб елемент `<tel> </tel>` був обов'язковим і вказувався декілька разів, всі інші елементи зробити необов'язковими.

2.2.2 Проект з XML Schema

Створити XML документ з прикладом резюме на основі завдання п. 2.2.1, але в якості правил розмітки документу використати мову XML Schema. Створити новий проект з ім'ям `task2`. Він повинен містити файли `resume.xml` та `xsd_resume.xsd`. Отримати повідомлення про успішну перевірку XML файлу.

Приклад генерації XSD з файлу XML у Liquid Studio (див. рис. 2.3). Потрібно зауважити, що при генерації кожен елемент має значення атрибуту `minOccurs` таким, що дорівнює 0. Після перегляду схеми і редагування атрибутів елементів, заміни деяких типів даних з типу `xs:string` на тип `xs:date` отримуємо текст файлу `xsd_resume.xsd` у наступному вигляді (див. лістинг 2.13). У файлі `resume.xml` вміст елементів з датами (рядки 7, 24, 25, 30, 31 лістингу 2.11) представляємо у вигляді PPPP-ММ-ДД та додаємо рядки коду для підключення XML Schema до XML (рядок 3 лістингу 2.14). Приклад виконання завдання у Liquid Studio (рис. 2.5).

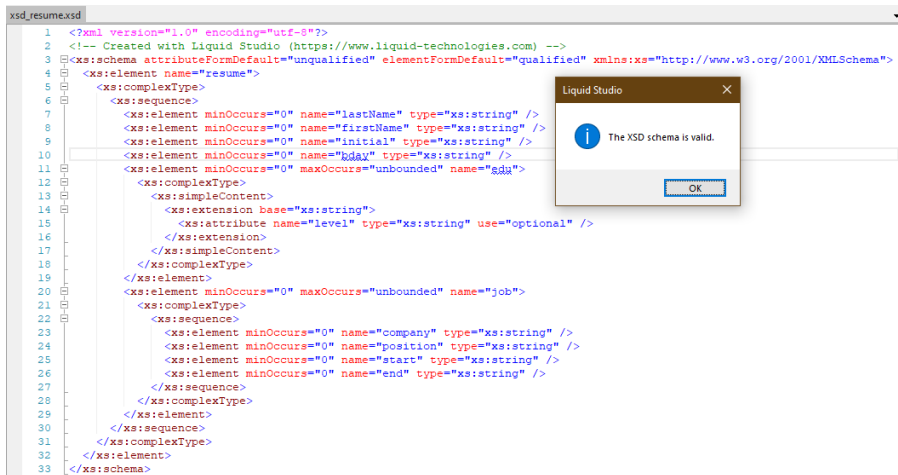


Рисунок 2.3 – Генерація XSD з файлу XML у Liquid Studio

Лістинг 2.13 Файл `xsd_resume.xsd`

```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <!-- Created with Liquid Studio
   (https://www.liquid-technologies.com) -->
3 <xs:schema
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
4   <xs:element name="resume">
5     <xs:complexType>
6       <xs:sequence>

```

```

7      <xs:element name="lastName"
          type="xs:string"/>
8      <xs:element name="firstName"
          type="xs:string"/>
9      <xs:element name="initial" type="xs:string"
          minOccurs="0"/>
10     <xs:element name="bday" type="xs:date"/>
11     <xs:element name="edu"
          maxOccurs="unbounded">
12         <xs:complexType>
13             <xs:simpleContent>
14                 <xs:extension base="xs:string">
15                     <xs:attribute name="level"
                          type="xs:string"
                          use="optional"/>
16                 </xs:extension>
17             </xs:simpleContent>
18         </xs:complexType>
19     </xs:element>
20     <xs:element name="job"
          maxOccurs="unbounded">
21         <xs:complexType>
22             <xs:sequence>
23                 <xs:element name="company"
                          type="xs:string"/>
24                 <xs:element name="position"
                          type="xs:string"/>
25                 <xs:element name="start"
                          type="xs:date"/>
26                 <xs:element name="end"
                          type="xs:date"/>
27             </xs:sequence>
28         </xs:complexType>
29     </xs:element>
30 </xs:sequence>
31 </xs:complexType>
32 </xs:element>
33 </xs:schema>

```

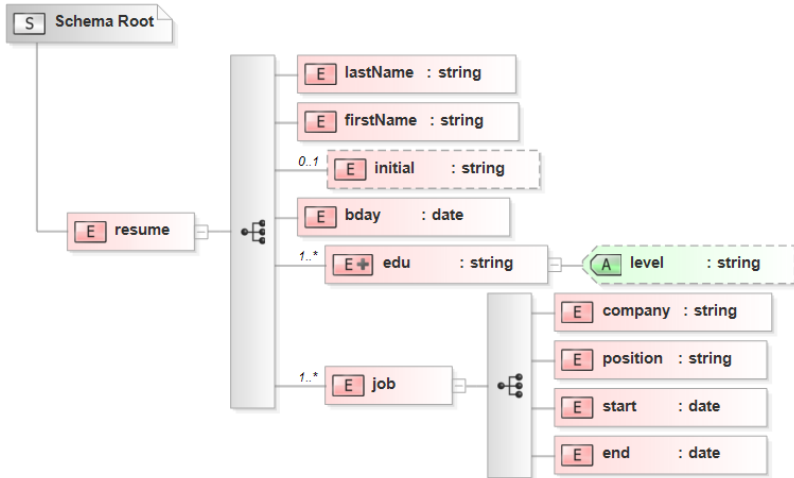


Рисунок 2.4 – Графічне зображення XSD файлу

Лістинг 2.14 Файл resume.xml проекту task2

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <!-- Created with Liquid Studio
      (https://www.liquid-technologies.com) -->
3  <resume xmlns:xsi=
      "http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="xsd_resume.xsd">
4      <lastName>Petryk</lastName>
5      <firstName>Pavlo</firstName>
6      <initial>Petrovych</initial>
7      <bday>2000-10-26</bday>
8      <edu level="school">
9          Zaporizhzhia Lyceum No.71
10     </edu>
11     <edu level="college">
12         Zaporizhzhia Vocational College of
13         Computer Technologies,
14         Computer engineering
15     </edu>
16     <edu level="university">

```

```

16      Zaporizhia Polytechnic National
      University, Computer Engineering
17    </edu>
18    <edu level="university">
19      National Aviation University,
      Cybersecurity and Information
      Protection
20    </edu>
21    <job>
22      <company>Freshcode</company>
23      <position>Java Developer</position>
24      <start>2021-11-25</start>
25      <end>2023-06-14</end>
26    </job>
27    <job>
28      <company>EPAM Ukraine</company>
29      <position>Java Developer</position>
30      <start>2023-06-24</start>
31      <end>2024-01-15</end>
32    </job>
33  </resume>

```

Завдання до проекту

Варіант 1. Доповнити XML документ таким чином, щоб тег `<company>` в інформації про роботу в елементі `<job>` `</job>` мав обов'язковий атрибут `type`, який може приймати значення “LTD” (*limited trade development*), “SOE” (*state-owned enterprise*) або “Public company” тощо.

Варіант 2. Доповнити XML документ можливістю додавати записи про контакти елементом `<contact>` `</contact>`. Тип контакту визначається атрибутом `type`, який може приймати значення `tel`, `address`, `email` або `messenger`. Зробити обов'язковим для введення від одного до десяти записів про контакти.

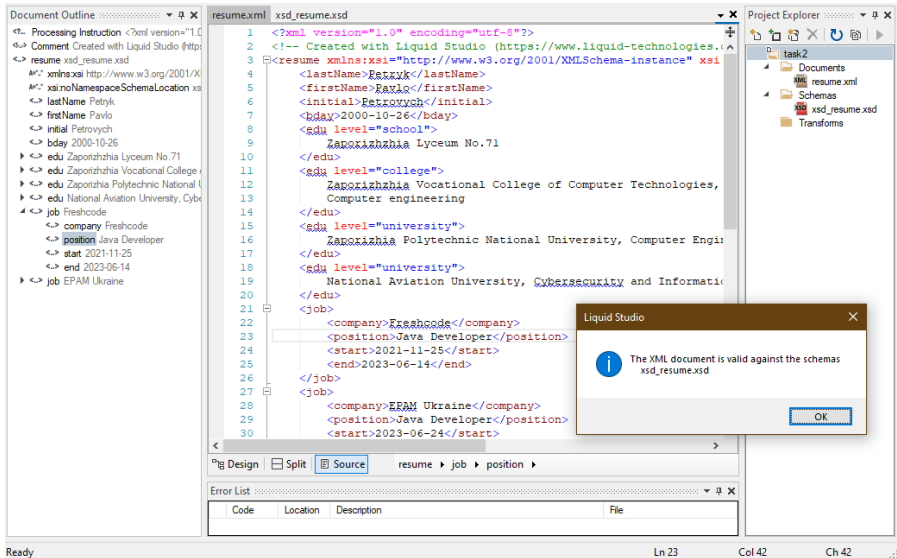


Рисунок 2.5 – Проект task2 у середовищі Liquid Studio

2.3 Зміст звіту

Звіт повинен містити: мету роботи, лістинги або фрагменти коду, що розроблені для власного прикладу програм, з необхідними поясненнями та відповіді на контрольні питання.

2.4 Контрольні питання

1. Дайте визначення XML.
2. Яким чином в XML задається підключення інших документів.
3. Чи обов'язковий заголовок в XML документі, яке він має значення.
4. Які види тегів існують в XML.
5. Назвіть основні синтаксичні правила XML.
6. Що таке DTD, для чого він використовується.
7. Яким чином підключити DTD до XML?

8. Як в DTD передається інформація про кількість частин елементів та необхідність їх використання?
9. Чи можливо використовувати XML без DTD?
10. Які типи даних підтримуються у XML Schema.
11. Наведіть приклад перетворення складного елемента у простий.
12. Яке призначення таблиці стилів XSL.
13. Які теги використовуються для створення шаблонів елементів з декількома значеннями атрибутів?

ЛАБОРАТОРНА РОБОТА 3

Практична робота в Java

Мета роботи — отримати навички роботи з пакетами, файлами, утилітами `ant` та `javadoc`.

Утиліта `ant` — це інструмент, призначений для автоматизації процесу збирання, тестування, налагодження та розгортання застосунків, написаних на Java, а також інших мовах програмування [7,8]. Утиліта `javadoc` призначена для автоматизації документування застосунків на мові Java [2,9].

3.1 Утиліта `ant`

Назва `ant` — акронім від Another Neat Tool. Інша версія користувачів: `ant` перекладається як “мураха”. Утиліта `ant` може застосовуватися як для маленьких програм, так і для великих проєктів, у тому числі при створенні Web-застосунків, розподілених програмних систем, до того ж `ant` є кросплатформним інструментом.

Утиліта `ant` обробляє XML-файл, у якому записуються завдання `ant`, тобто завдання, які має виконати `ant` для цього проєкту. Цей файл створюється та редагується програмістом і називається *build-файлом*. Кожен build-файл призначений для одного проєкту і містить одне або декілька *завдань (целей, target)*. Кожна ціль може містити одну або декілька *задач (task)*.

На рис. 3.1 показана структура простого проєкту, який виконуватиметься за допомогою утиліти `ant`.

3.1.1 Програма “Паліндром”

Для практичного вивчення утиліти `ant` виконайте наступне:

1. Створіть у робочому каталозі папку `antlab`, у якій створіть ієрархію папок `src\edu\nuzp\username`;
2. У папку `antlab` помістіть файл `build.xml`, текст якого наведений у лістингу 3.1;

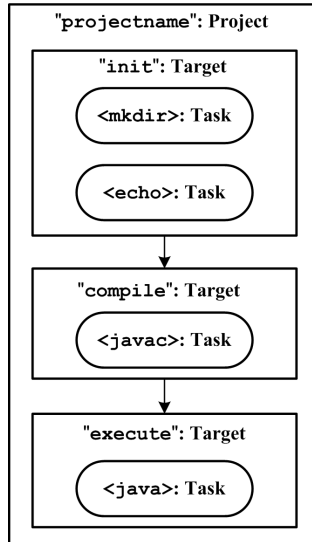


Рисунок 3.1 – Структура build-файлу

3. У папку **username** помістіть файл **DemoAdamAndEva.java**, код якого наведено на лістингу 3.2;

4. У командному рядку перейдіть до папки **antlab** і виконайте команду **ant**. В результаті на екрані виведуться рядки, показані на лістингу 3.3.

Лістинг 3.1 Файл build.xml для програми "Паліндром"

```

1  <?xml version="1.0"?>
2  <project name="ProjectPalindrom"
    default="execute">
3
4      <target name="init">
5          <mkdir dir="build"/>
6          <echo>init complete!</echo>
7      </target>
8
9      <target name="compile" depends="init" >
10         <javac srcdir="src" destdir="build"
  
```

```

        includeantruntime="false"/>
11     <echo>compilation complete!</echo>
12 </target>
13
14 <target name="execute" depends="compile">
15     <java
        classname="edu.nuzp.username.DemoAdamAndEva"
        classpath="build">
16     </java>
17 </target>
18 </project>

```

Лістинг 3.2 Програма "Паліндром"

```

1  package edu.nuzp.username;
2
3  class Adam{
4      void heSaid(){
5          System.out.println("Madam I'm Adam");
6          System.out.println("it is palindrome:
           madamimadam");
7      }
8  }
9
10 class Eva {
11     void sheAnswered(){
12         System.out.println("Hello!");
13     }
14 }
15
16 public class DemoAdamAndEva {
17     public static void main(String args[]) {
18         Adam Adam = new Adam ();
19         Eva eva = new Eva();
20         adam.heSaid();
21         eva.sheAnswered();
22     }
23 }

```

Лістинг 3.3 Результат роботи програми "Паліндром"

```
init:
[echo] init complete!

compile:
[javac] Compiling 1 source file to
      D:\JavaPrj\antlab\build
[echo] compilation complete!

execute:
[java] Madam I'm Adam
[java] it is palindrome: madamimadam
[java] Hello!

BUILD SUCCESSFUL
Total time: 1 second
```

Пояснимо роботу утиліти **ant** з файлом **build.xml** (див. лістинг 3.1). У рядку 2 вказано ім'я проекту, яке не відіграє ніякої ролі, та завдання (**target**), яке виконується у проекті за замовчуванням (**default** = "**execute**"). Отже, завданням є виконання програми. У свою чергу завдання "**execute**" (рядок 14) залежить від завдання компіляції **depends="compile"** (рядок 9), а завдання компіляції залежить від завдання ініціалізації **depends="init"** (рядок 4), яке має завдання створити каталог **<mkdir dir="build"/>**. Отже, утиліта **ant** спочатку виконає завдання **init**, потім завдання **compile** і, нарешті, завдання **execute**.

Під час виконання завдань створення каталогів та компіляції **ant** завжди перевіряє, чи вносилися зміни у вихідний файл програми та чи є необхідні каталоги. Якщо зміни до вихідного файлу не вносилися, то він не виконуватиме компіляцію (завдання буде пропущено), і якщо каталоги вже існують, то вони не будуть створюватися.

Кожне із завдань може бути виконане окремо. Наприклад, команда **ant mkdir** викликає лише створення каталогу. Завдання можуть бути без залежностей, тобто виконання цього завдання не залежить від інших. Прикладом може бути завдання, в результаті

якого **ant** вилучить каталог **build**, якщо виконати команду **clean**:

```
<target name="clean">
  <echo>cleaning up</echo>
  <delete dir="build"/>
</target>
```

Для введення параметрів з командного рядка у **build**-файлі у завдання **"execute"** слід включити рядок **<arg value="\$param"/>**

```
<target name="execute" depends="compile">
<java...>
  <arg value="{param}"/>
</java>
</target>
```

та виконати наступну команду:

```
ant -Dparam=AnyString
```

де **AnyString** — параметр командного рядка Java-програми.

Завдання до програми

1. Вивчіть та реалізуйте програму з лістингів 3.1—3.3. Продемонструйте її працездатність, виконуючи команди у командному рядку;

2. Доопрацюйте **build**-файл так, щоб у повідомленнях утиліти **ant** були назва проекту, ім'я та група студента;

3. Доопрацюйте **build**-файл та Java-програму так, щоб:

- програма розміщувалася в пакеті **myantlab**, а Java-файл у папці **src\edu\nuzp\username**, де **username** — ім'я студента;

- **build**-файл повинен дозволити вводити параметри командного рядка, наприклад ім'я студента користувача;

- Java-програма повинна виводити на друк параметри командного рядка;

- у **build**-файл ввести завдання, за допомогою якого з командного рядка можна видаляти каталог **build**.

3.2 Файлове введення-виведення у Java

Будь-які серйозні програми повинні вміти взаємодіяти із зовнішніми системами, приймаючи від них дані та надаючи результати обробки даних. Технологія Java надає потужну та гнучку підтримку файлового введення-виводу та взаємодії з файловими системами. Для цього у Java є ієрархія класів потоків введення-виводу, що об'єднує більш ніж 60 типів потоків (для порівняння в C++ всього 18 типів потоків) [2]. Ми на прикладах розглянемо лише базові класи та їх методи для запису даних у файл та зчитування даних із файлу.

Потоками вводу-виводу Java називаються *об'єкти*, в які та з яких записуються/зчитуються послідовності даних.

Для *символьного* введення даних із файлу з підвищеною продуктивністю зручно використовувати клас **BufferedReader**, ініціалізований об'єктом класу **FileReader**, який має метод посимвольного читання **read()** та рядкового читання **readLine()**.

Для *символьного* виведення даних у файл слід використовувати методи класу **PrintWriter**, який ініціалізується об'єктом класу **BufferedWriter**, ініціалізованого, у свою чергу, об'єктом класу **FileWriter**. Клас **PrintWriter** має багато методів для керування виведенням даних у файл, у тому числі перевантажені методи **print()** і **println()** для виведення різних типів даних, а також методи форматowanego виведення даних **printf()** та **format()**.

Для зв'язування потоків із конкретними файлами є клас **File**. Методи класу **File** призначені для управління файлами та каталогами, отримання інформації про файли (шлях, розмір, час та дата модифікації тощо).

3.2.1 Програма роботи з файлами

У даній програмі (лістинг 3.4) продемонстровані прийоми роботи з символьними потоками, які часто використовуються на практиці, у тому числі клас **Scanner**, методи якого призначені для зчитування різних даних (символів, слів, рядків, цифр) з вхідного потоку.

Для організації читання даних з консолі у конструктор класу `Scanner` передається об'єкт стандартного вхідного потоку `System.in`.

Лістинг 3.4 Програма роботи з файлами

```

1 package edu.nuzp.file;
2
3 import java.io.*;
4 import java.text.SimpleDateFormat;
5 import java.util.Scanner;
6
7 public class DemoFile {
8     public static void main(String[] args){
9         Scanner consoleIn = New Scanner (System.in);
10        System.out.println("Hello, enter your
11        name");
12        String name = consoleIn.nextLine();
13        System.out.println("Enter your academical
14        group");
15        String group = consoleIn.nextLine();
16        consoleIn.close();
17        String text = name + " " + group + "\n";
18        String filename = "myinfo.txt";
19        PrintWriter writer = null;
20        BufferedReader reader = null;
21        File file = new File(filename);
22        try{
23            writer =new PrintWriter(new
24                BufferedWriter(new FileWriter(file)));
25            writer.println(text);
26            if(writer != null){
27                writer.close();
28            }
29            reader = new BufferedReader(new
30                FileReader(file));
31            String line;
32            while((line = reader.readLine()) !=
33                null){
34                System.out.println(line);

```

```

30         }
31         if(reader != null){
32             reader.close();
33         }
34     } catch (IOException ex) {
35         System.out.println(ex.getMessage());
36     }
37     if(file.exists()){
38         if(file.isFile()){
39             System.out.println("filename:\t" +
40                 file.getName());
41             System.out.println("path:\t " +
42                 file.getAbsolutePath());
43             System.out.println("length:\t" +
44                 file.length());
45             SimpleDateFormat sdf = new
46                 SimpleDateFormat("dd.MM.yyyy
47                     HH:mm:ss");
48             System.out.println("modified:\t " +
49                 sdf.format(file.lastModified()));
50         }
51     } else
52         System.out.println("file" + file.getName()
53             + "is missing");
54 }
55 }
```

Завдання до програми

1. Вивчіть та реалізуйте програму з лістингу 3.4, при цьому за необхідності внесіть зміни до оператора **package**;
2. Розробіть власну програму, яка зчитує текст із існуючого текстового файлу (для цього створіть файл в текстовому редакторі Блокнот) та записує зчитаний текст у новий текстовий файл;
3. Доопрацюйте розроблену програму так, щоб перший рядок записуваного тексту містив назву файлу, дату та час створення файлу, а також ім'я студента.

3.3 Документування коду

Коментарі, які починаються з символічної послідовності `/**` і закінчуються послідовністю `*/`, можуть документуватися за допомогою утиліти `javadoc`, яка отримує ці коментарі та поміщає їх у HTML-файли. Ці файли є зручними поясненнями до програми. Рекомендується використовувати ці коментарі для роз'яснення класів, інтерфейсів, полів, конструкторів та методів.

Коментарі для документування повинні розташовуватися *безпосередньо* перед елементами, що коментуються. Утиліта `javadoc` отримує інформацію про наступні елементи коду програми:

- пакети;
- класи, оголошені як `public`;
- інтерфейси;
- методи, оголошені як `public` або `protected`;
- поля, оголошені як `public` або `protected`.

Коментарі можуть бути як загального, описового характеру, так і позначені спеціальними дескрипторами, які починаються із символу `@`. Найчастіше використовуються такі дескриптори:

- `@author ім'я` – задає відомості про автора коду;
- `@version текст`, тут слово *текст* – довільний опис поточної версії фрагмента коду (наприклад, класу);
- `@since` – визначає версію (наприклад `jdk1.8`) або час, починаючи з якого даний код є актуальним;
- `@see посилання` – додає посилання в розділ "`See also...`", тобто, що слід подивитися додатково;
- `@param змінна опис`, описується параметр, що передається в метод;
- `@return опис` – описує значення методу, що повертається;
- `@exception` – задає ім'я класу виключення;
- `@throws` – визначає виключення, що генерується методом;
- `@deprecated` – попередження про те, що метод застарів і має більш досконалий аналог.

3.3.1 Програма для обчислення числа Ейлера

Програма (лістинги 3.5, 3.6, 3.7) реалізує застосунок, у якому обчислюється число Ейлера за формулою $e = (1 + 1/n)^n$. Оскільки число Ейлера є границею цього виразу при $n \rightarrow \infty$, точність обчислення залежатиме від значення n . Обчислення виконуються для $n = 10, 100, \dots, 10^8$. Результати обчислення записуються у файл у робочу папку проекту. Потім програма зчитує вміст цього файлу і виводить на консоль.

Ця програма призначена для демонстрації деяких прийомів роботи з файлами та забезпечена прикладами документованих коментарів, з яких очевидним є призначення коментованих ділянок коду.

Лістинг 3.5 Клас для обчислення числа Ейлера

```

1  package edu.nuzp.doc;
2
3  import java.util.Locale;
4  /**
5   * This class contain a method for Euler's number
      calculation
6   * @author Kudermetov R.
7   * @version 1.1
8   * @since 2024-10-29
9   */
10
11 public class Exponent {
12
13     private String string = new String();//Empty
      string
14     /**
15      * This method calculate Euler's number
      depending on
16      * <em> i </em>, which changes as 10, 100, ...,
      100000000.
17      * Method writes the results of calculating to
      string by
18      * <code> String.format() </code> method.
```

```

19      * @return string
20      * @see <a href="https://docs.oracle.com/javase/
21      7/docs/api/java/lang/String.html">
        String.format() </a>
22      */
23      public String calcExponent(){
24          for(int i = 10; i < 100_100_101; i * = 10) {
25              double e = Math.pow((1.0 + 1.0/i), i);
26              string += String.format(Locale.ENGLISH,
                "i = %d, \ty = %9.8f\n", i, e);
27          }
28          return string;
29      }
30  }

```

Лістинг 3.6 Клас для роботи з файлами

```

1  package edu.nuzp.doc;
2
3  import java.io.*;
4  /**
5   * This class contains the methods for working
        with file.
6   * @author Polska O.
7   * @version 1.0
8   * @since 2024-10-30
9   */
10 public class InOutFile {
11     private File file;
12
13     /**
14      * Constructor which creates File class object
        by
15      * initializing it with the file name.
16      * @param string is the name of the file on the
        disk
17      */
18
19     public InOutFile(String string){
20         file = New File (string);

```

```

21     }
22
23     /**
24     * This method writes content with calculated
25         Euler's
26     * numbers to file.
27     * @param text content for writing to file
28     * @see java.io.IOException
29     */
30     public void writer(String text) {
31         try{
32             PrintWriter pw = new PrintWriter(new
33                 BufferedWriter(new FileWriter(file)));
34             pw.println(text);
35             if(pw != null){
36                 pw.close();
37             }
38         } catch (IOException ex) {
39             System.out.println(ex.getMessage());
40         }
41     }
42
43     /**
44     * This method reads content of file.
45     * @return content of file
46     * @see java.io.IOException
47     */
48     public String reader() {
49         String text = new String();
50         String line = new String();
51         try{
52             BufferedReader br = New
53                 BufferedReader(New FileReader(file));
54             while((line = br.readLine()) != null){
55                 text += line + "\n";
56             }
57             if (br! = null) {
58                 br.close();
59             }
60         } catch (IOException ex) {

```

```

58         System.out.println(ex.getMessage());
59     }
60     return text;
61 }
62 }

```

Лістинг 3.7 Головний клас програми

```

1  package edu.nuzp.doc;
2  /**
3   * This is a program for laboratory work in
4   * discipline
5   * Modern Methods of Programming. The DemoDoc
6   * program
7   * implements an application that calculates
8   * Euler's number
9   * as limit of expression  $(1+1/n)^n$ . The
10  * results
11  * of calculation depending on values  $n$ 
12  * are written
13  * to a file, then are read from the file and
14  * output
15  * to a console.
16  * @author Kudermetov, R.
17  * @author Shkarupylo, V.
18  * @author Polska, O.
19  * @version 1.0
20  * @ Since 2024-11-05
21  */
22  public class DemoDoc {
23  /**
24   * This is main() method which creates objects of
25   * classes
26   * Exponent and InOutFile. It calls the method of
27   * calculation
28   * Euler's number, and the methods for writing and
29   * reading
30   * results to/from file.
31   * @param args unused
32   * @see edu.nuzp.doc.Exponent

```

```

24 * @see edu.nuzp.doc.InOutFile
25 */
26 public static void main(String[] args){
27     String str="exponent.txt";
28     InOutFile iof = new InOutFile(str);
29     Exponent exp = new Exponent();
30     iof.writer(exp.calcExponent());
31     System.out.println(iof.reader());
32 }
33 }

```

Завдання до програми

1. Вивчіть та реалізуйте програму з лістингів 3.5—3.7. Для цього можуть знадобитися зміни в назвах пакетів (оператор **package**);
2. Змініть дані в коментарях, щоб вони відповідали імені студента, версії класів та даті реалізації;
3. Згенеруйте у відповідному документі IDE додаток (в IDE Eclipse: "Project → Generate Javadoc..."), при цьому:
 - переконайтеся, що у вікні консолі IDE утиліта **javadoc** не повідомляє про помилки;
 - у браузері перевірте наявність всіх коментованих дескрипторів та відповідних їм значень на сторінках згенерованої документації (вона повинна знаходитись у папці каталогу проекту **doc**);
4. Створіть ієрархію папок **antdoc\src\edu\nuzp\username** у робочому каталозі, де **username** – ім'я студента-користувача, і помістіть у *цю* папку файли із вихідним кодом програми;
5. Внесіть необхідні зміни до назв пакетів в операторах **package** вихідних файлів проекту;
6. Внесіть необхідні зміни у посиланнях директив **@see** (назви пакетів);
7. Розробіть **build**-файл, в якому, на відміну від лістингу 3.1, має бути додаткова мета **"doc"**. Наприклад:

```

<target name="doc" depends="compile" >
    <javadoc destdir="doc" sourcepath="src"
        packagenames="edu.*" />
</target>

```

Мета "init" повинна включати наступне завдання:

```
<mkdir dir="doc" />
```

Мета "execute" повинна залежати від цілей "compile" і "doc":

```
<target name="execute" depends="compile,doc">
```

У цілі "execute" в атрибуті `classname` потрібно вказати правильне кваліфіковане ім'я;

8. Включіть у цілі "init" завдання виведення на консоль назви проекту та імені студента;

9. Збережіть створений файл `build.xml` у папці `antdoc`;

10. Продемонструйте створення програми, документації до неї та її виконання за допомогою утиліти `ant`;

11. Доопрацюйте програму так, щоб у файл додатково записувалися похибки обчислення числа Ейлера для різних значень n , взявши за точне значення $e = 2,71828182845904523536$.

12. Додайте в документований коментар методу `calcExponent()` пояснення про його доопрацювання.

13. Продемонструйте працездатність доопрацьованої програми за допомогою утиліти `ant`.

14. Додайте у `build`-файл незалежну мету "clean" для видалення каталогів `build` та `doc`;

15. Видаліть за допомогою команди `ant clean` каталоги `build` та `doc`;

16. Збережіть повідомлення утиліти `ant` у файл `info.txt`, виконавши у командному рядку команду `ant > info.txt`. У звіті до лабораторної роботи наведіть вміст цього файлу.

3.4 Зміст звіту

Звіт повинен містити: мету роботи, лістинги або фрагменти коду, що розроблені для власного прикладу програм, з необхідними поясненнями та відповіді на контрольні питання.

3.5 Контрольні питання

1. Призначення і принцип роботи утиліти **ant**.
2. Які завдання можна виконати за допомогою утиліти **ant**.
3. Напишіть, як за допомогою утиліти **ant** створити каталог.
4. Напишіть, як за допомогою утиліти **ant** видалити каталог.
5. Які засоби забезпечення файлового введення-виведення у мові Java ви знаєте.
6. Які методи та класи Java ви використовували для роботи з символьними даними.
7. Призначення і принцип роботи утиліти **javadoc**.
8. Про які елементи коду програми можна отримати інформацію за допомогою утиліти **javadoc**.
9. Які спеціальні дескриптори утиліти **javadoc** ви знаєте.
10. Які спеціальні дескриптори утиліти **javadoc** ви використовували в даній лабораторній роботі.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

- [1] Horstmann C. S. *Core Java. Volume I: Fundamentals. 13-th ed.* Pearson Education, 2024.
- [2] Horstmann C. S. *Core Java. Volume II: Advanced Features. 13-th ed.* Pearson Education, 2024.
- [3] Extensible Markup Language (XML) 1.1 (Second Edition). URL: <https://www.w3.org/TR/2006/REC-xml11-20060816/>.
- [4] XML Tutorial – W3Schools. URL: <https://www.w3schools.com/xml/default.asp>.
- [5] NetBeans XML Editor Extension Module Tutorial. URL: <https://netbeans.apache.org/tutorial/main/tutorials/nbm-xmleditor/>.
- [6] Liquid Studio - toolkit for XML and JSON development. URL: <https://www.liquid-technologies.com/xml-studio>.
- [7] Apache Ant - Welcome. URL: <https://ant.apache.org/>.
- [8] Loughran S. and Hatcher E. *Ant in Action: Second Edition of Java Development with Ant.* Manning Publications Co., 2007.
- [9] How to Write Doc Comments for the Javadoc Tool. URL: <https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html>.