

Міністерство освіти і науки  
Національний університет «Львівська політехніка» Інститут комп'ютерних наук  
та інформаційних технологій Кафедра програмного забезпечення



**Звіт**  
**із виконання лабораторної роботи № 3 «Робота з текстовою інформацією,**  
**файлами та серіалізація.»**  
**з дисципліни «Кросплатформне програмування»**

**Лектор:**

доцент кафедри ПЗ Дяконюк Л.М.

**Виконав:**

студент групи ПЗ-36 Баш І. О.

**Прийняв:**

асистент кафедри ПЗ

Баштовий А.В.

«\_\_» \_\_\_\_\_ 2024 р.  $\Sigma$  = \_\_\_\_\_

**Тема:** робота з текстовою інформацією, файлами та серіалізація.

**Мета:** навчитися працювати з потоками введення-виведення (I/O Streams), текстовою інформацією, файлами, нативною серіалізацією та сторонніми бібліотеками для серіалізації об'єктів у формати JSON та YAML. Ознайомитися з використанням Maven для управління залежностями та збіркою проекту, вивчити особливості використання систем збірки проектів.

## Завдання

### 1. Збірка проектів за допомогою Maven:

- Створіть Maven проект.
- Додайте залежності у файл `pom.xml` для бібліотек **Gson**, **Jackson**, та **SnakeYAML** або альтернативні які потрібні для завдань нижче.
- Налаштуйте проект так, щоб використовувати ці бібліотеки для серіалізації/десеріалізації.
- Виконайте збірку проекту за допомогою команди `mvn clean install` і переконайтеся, що проект компілюється та працює без помилок.

### 2. Робота з Java I/O Streams:

- Використайте:
  - Небуферизовані потоки (`FileInputStream`, `FileOutputStream`) для читання та запису байтів.
  - Буферизовані потоки (`BufferedInputStream`, `BufferedOutputStream`) для оптимізації процесу читання та запису.
  - Використайте `ObjectInputStream`

### 3. Серіалізація та десеріалізація об'єктів Java:

- Створити клас з кількома полями одне з яких інший клас, а не примітивний тип.
- Напишіть програму, яка:
  - Створює об'єкт цього класу, серіалізує його в файл
  - Читає з цього файлу та десеріалізує об'єкт
- Вивести серіалізовані дані та результат десеріалізації

### 4. Серіалізація JSON :

- Використовуючи бібліотеку **Gson** або **Jackson** або альтернативну, одну з них на вибір:
  - Серіалізуйте об'єкт цього класу у формат JSON та запишіть його у файл.
  - Прочитайте JSON файл, десеріалізуйте його назад в об'єкт Java та виведіть на екран.
- Переконайтеся, що всі поля класу правильно серіалізовані та десеріалізовані.

## 5. Серіалізація YAML:

Використовуйте бібліотеку **SnakeYAML** або альтернативну для:

- Серіалізації об'єкта Java у YAML файл.
- Читання YAML файлу та десеріалізації його в об'єкт Java.
- Виведіть об'єкт, що був десеріалізований, для перевірки його коректності.

## Варіант 2: Облік товарів в інтернет-магазині

### 1. Робота з потоками вводу-виводу:

- Читання та запис інформації про товари (назва, ціна, кількість, список постачальників) у файл.
- Список постачальників містить назву компанії та дату початку контракту.
- **Буферизовані потоки** використовуйте для постачальників, а **небуферизовані** для запису товарів.
- **Умова:** Поле `ціна` товару **не повинно бути серіалізоване**.

### 2. Нативна серіалізація Java:

- Серіалізуйте список товарів у файл.
- **Умова:** Список постачальників серіалізувати **не потрібно**.

### 3. Серіалізація JSON (GSON або Jackson):

- Серіалізуйте товари у JSON, але поле `ціна` **не повинно бути присутнє** в результаті серіалізації.
- Читання JSON має відновити всі інші поля.

### 4. Серіалізація YAML (SnakeYAML):

- Серіалізуйте товари у YAML, але **пропустіть поле з датою початку контракту** для постачальників.

## Теоретичні відомості

### 1. I/O Streams (Вводу-Виводу потоки):

- Потоки вводу-виводу (I/O Streams) в Java використовуються для читання та запису даних з різних джерел, таких як файли, мережа або системні ресурси. Існують буферизовані потоки, які зменшують кількість фізичних операцій читання/запису, та небуферизовані, які працюють напямую з джерелом даних.
- [Документація Java I/O Streams](#)
- [Офіційна документація Maven](#)

### 2. JSON (JavaScript Object Notation):

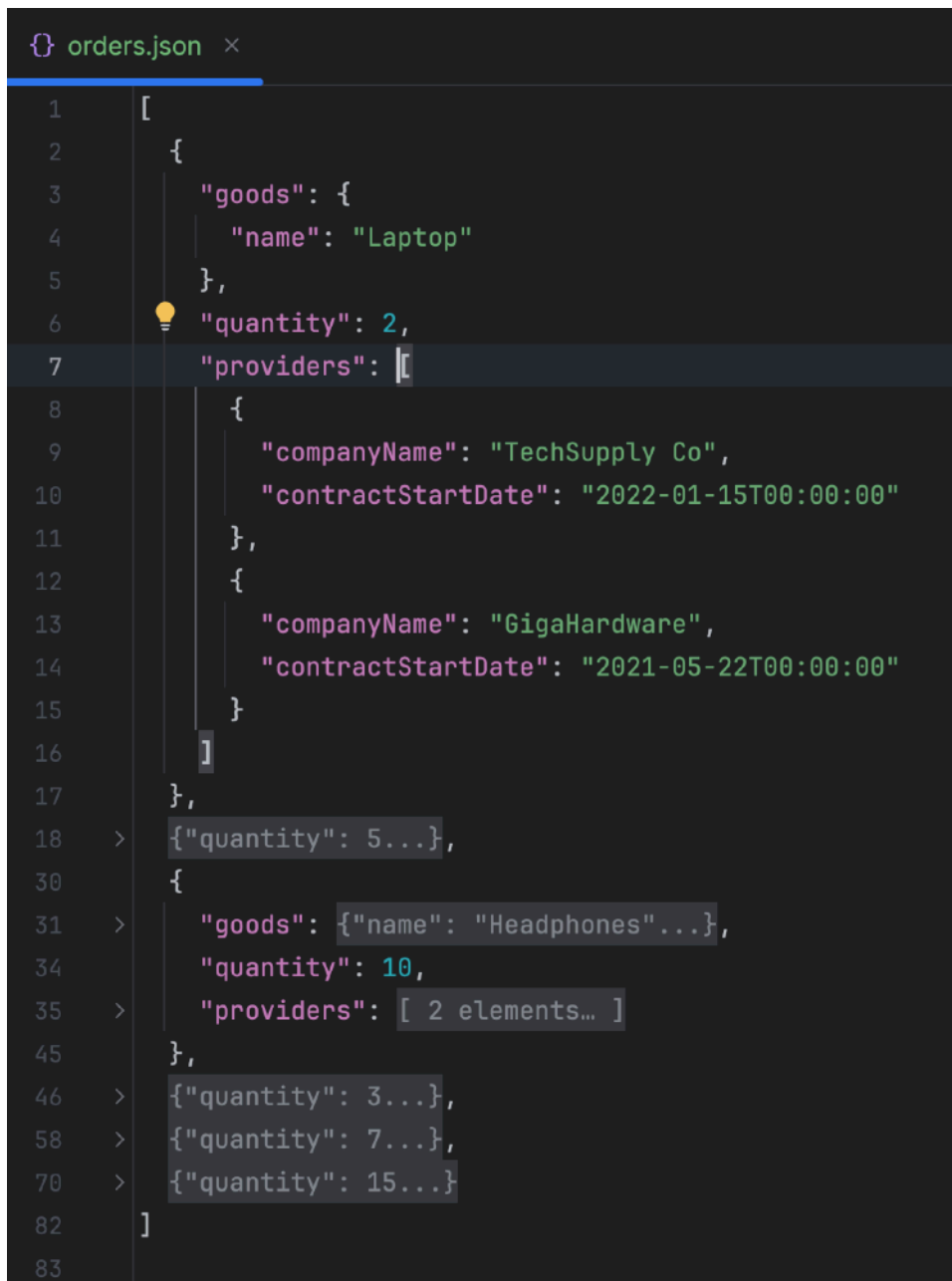
- JSON – це легкий формат обміну даними, який зручно використовувати для передачі структурованих даних між сервером і клієнтом. Він широко застосовується в веб-розробці для серіалізації об'єктів.
- Документація JSON

### 3. YAML (YAML Ain't Markup Language):

- YAML – це простий у використанні формат серіалізації даних, що підтримує читання та редагування людиною. Він часто використовується для конфігураційних файлів завдяки своїй читабельності та гнучкості.
- [Документація YAML](#)

**Код програми:** [https://gitlab.com/ivan.bash.pz.2022/cpp/-/tree/main/Lab3/src/main?ref\\_type=heads](https://gitlab.com/ivan.bash.pz.2022/cpp/-/tree/main/Lab3/src/main?ref_type=heads)

### Результат виконання програми



```
orders.json x
1  [
2    {
3      "goods": {
4        "name": "Laptop"
5      },
6      "quantity": 2,
7      "providers": [
8        {
9          "companyName": "TechSupply Co",
10         "contractStartDate": "2022-01-15T00:00:00"
11       },
12       {
13         "companyName": "GigaHardware",
14         "contractStartDate": "2021-05-22T00:00:00"
15       }
16     ],
17   },
18   {"quantity": 5...},
30   {
31     "goods": {"name": "Headphones"...},
34     "quantity": 10,
35     "providers": [ 2 elements... ]
45   },
46   {"quantity": 3...},
58   {"quantity": 7...},
70   {"quantity": 15...}
82 ]
83
```

Рис. 1. Файл з початковими даними



Рис. 2. Файли створені після виконання серіалізації

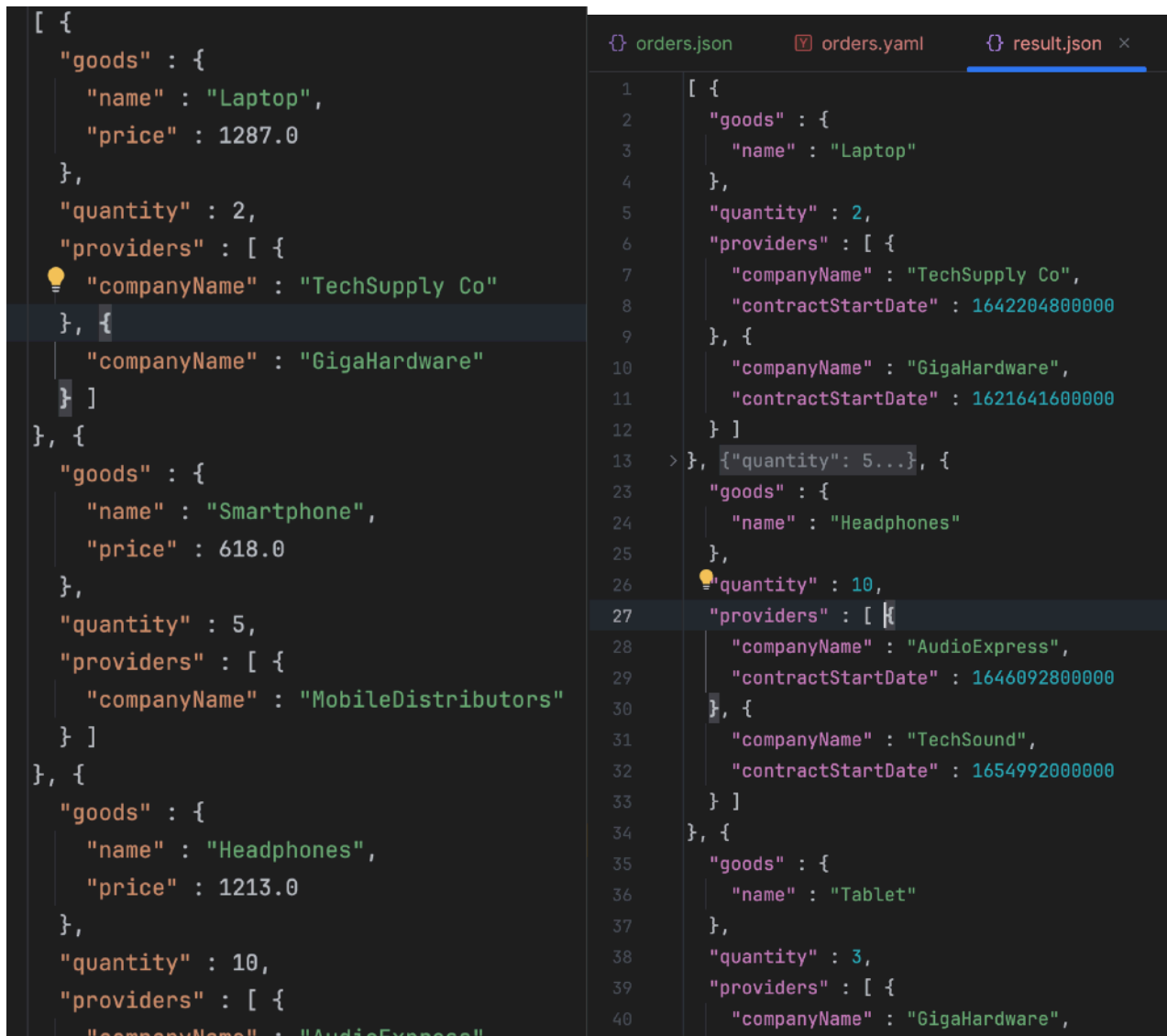


Рис. 3. Вміст файлів створених після виконання серіалізації

**Код тестів:** [https://gitlab.com/ivan.bash.pz.2022/cpp/-/tree/main/Lab3/src/test/java?ref\\_type=heads](https://gitlab.com/ivan.bash.pz.2022/cpp/-/tree/main/Lab3/src/test/java?ref_type=heads)

✓ <default package>	250 ms
> ✓ NativeSerializerTest	42 ms
> ✓ ProviderWriterTest	37 ms
> ✓ ProviderReaderTest	1 ms
> ✓ NativeDeserializerTest	6 ms
✓ JacksonReaderTest	133 ms
✓ testReadList()	127 ms
✓ testReadSingleOrder()	6 ms
> ✓ GoodsWriterTest	
> ✓ GoodsReaderTest	
> ✓ JacksonSerializatorTest	11 ms
✓ YamlSerializerTest	20 ms
✓ testDumpAllOrders()	19 ms
✓ testSerializeSingleOrder()	1 ms
✓ testDumpSingleOrder()	

*Рис. 4. Результат виконання тестів*

### Висновки

У результаті виконання лабораторної роботи № 3 «Робота з текстовою інформацією, файлами та серіалізація» були досягнуті важливі цілі, які суттєво підвищили рівень знань і практичних навичок. Під час роботи студенти ознайомилися з використанням небуферизованих (`FileInputStream`, `FileOutputStream`) та буферизованих потоків (`BufferedInputStream`, `BufferedOutputStream`), що дозволило оптимізувати процес читання та запису даних.

Крім того, було реалізовано серіалізацію об'єктів Java за допомогою нативної серіалізації, а також у форматах JSON і YAML, використовуючи бібліотеки `Gson` і `SnakeYAML`. Важливим етапом стало створення Maven проекту, що підтвердило значення системи управління проектами для організації роботи.

Отримані знання знайшли практичне застосування в розробці програми для обліку товарів в інтернет-магазині, де були реалізовані функції читання та запису інформації, а також серіалізації. Загалом, лабораторна робота сприяла закріпленню розуміння роботи з текстовою інформацією, файлами та серіалізацією в Java, що є корисним для подальшого навчання і професійного розвитку.