Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 7

з дисципліни «**Технології розроблення програмного забезпечення**»

Патерни проектування.

Виконав:

Студент групи ІА-31

Шереметьєв Дмитро

Київ – 2025

Вихідний код: https://github.com/Dimon4ick68/MindMapLabs

Тема: Патерни проектування.

Мета: Вивчити структуру шаблонів «Mediator», «Facade», «Bridge», «Template method» та навчитися застосовувати їх в реалізації програмної системи.

Завдання

Ознайомитись з короткими теоретичними відомостями.

• Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

• Реалізувати один з розглянутих шаблонів за обраною темою.

• Реалізувати не менше 3-х класів відповідно до обраної теми.

• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

Варіант:

20. **Mind-mapping software** (strategy, prototype, abstract factory, bridge, composite, SOA)

Візуальний додаток для складання "карт пам'яті" з можливістю роботи з декількома картами (у вкладках), автоматичного промальовування ліній, додавання вкладених файлів, картинок, відеофайлів (попередній перегляд); можливість додавання значків категорій / терміновості, обведення областей карти (поділ пунктирною лінією).

Хід роботи

У рамках виконання лабораторної роботи було реалізовано підсистему експорту даних для редактора ментальних карт. Для цього використано структурний патерн **Bridge (Міст)**, який дозволяє розділити високорівневу логіку експорту (що саме ми експортуємо і як обходимо карту) від низькорівневої реалізації форматування (в який саме формат записуємо дані).

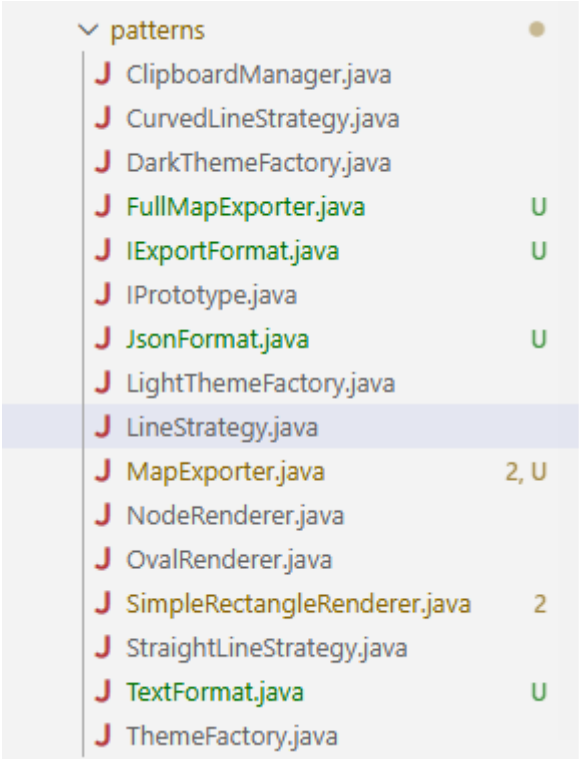


Рисунок 1 Структура проекту

Реалізовано такі компоненти патерну:

1. IExportFormat (Implementor — Реалізація) Інтерфейс, що оголошує базові операції форматування даних, незалежні від структури самого документа.

- **Методи:**
    - formatHeader(map): повертає відформатований заголовок файлу.
    - formatNode(node, level): повертає рядкове представлення окремого вузла з урахуванням рівня вкладеності.
    - formatFooter(map): повертає підвал файлу.

2. TextFormat та JsonFormat (Concrete Implementors — Конкретні реалізації) Класи, що реалізують інтерфейс IExportFormat. Кожен клас відповідає за специфіку конкретного формату файлу:

- **Текстовий формат (TextFormat):** Формує простий список, використовуючи символи табуляції для візуалізації ієрархії дерева.
- **JSON формат (JsonFormat):** Використовує бібліотеку серіалізації для створення валідних JSON-об'єктів, придатних для машинної обробки.

3. MapExporter (Abstraction — Абстракція) Базовий клас експортера, який містить посилання на об'єкт типу IExportFormat (сам "міст"). Він визначає загальний алгоритм експорту (відкриття файлу, запис), делегуючи детальне форматування об'єкту реалізації. Клієнт взаємодіє з системою експорту саме через цей клас.

4. FullMapExporter (Refined Abstraction — Уточнена абстракція) Конкретний клас експортера, що розширює абстракцію. Він реалізує логіку повного рекурсивного обходу дерева вузлів ментальної карти. Для кожного відвіданого вузла він викликає метод formatNode у підключеного форматера, не знаючи деталей реалізації формату.

5. EditorForm (Client — Клієнт) Клас редактора, який ініціює процес експорту. У момент виклику функції експорту він "збирає міст": створює об'єкт абстракції (FullMapExporter) і передає йому потрібну реалізацію (JsonFormat або TextFormat) залежно від вибору користувача у діалоговому вікні.

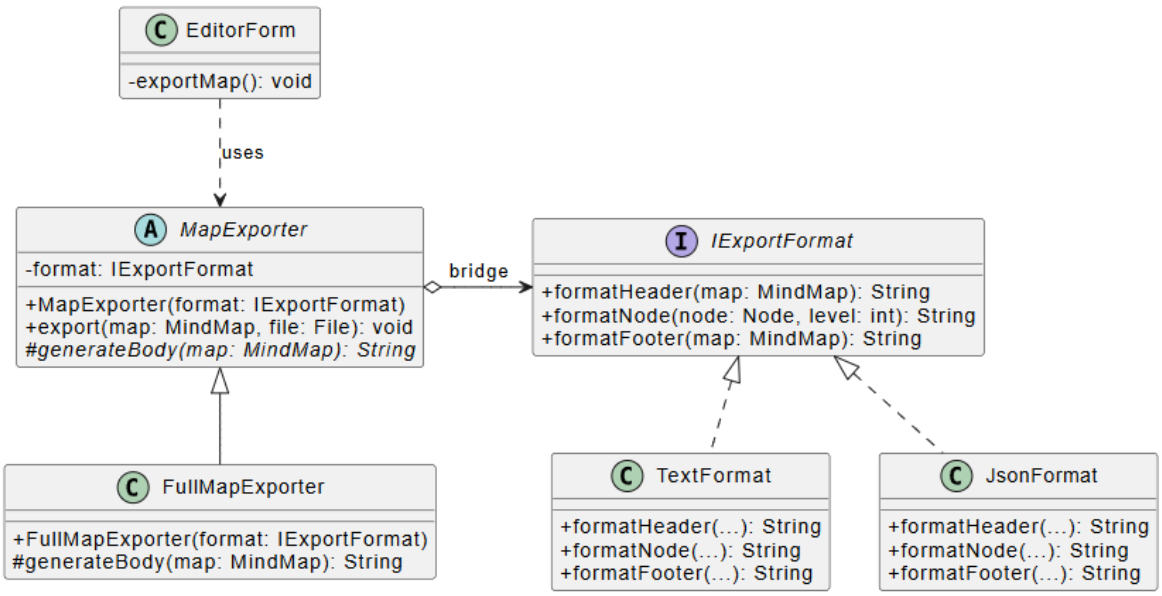


Рисунок 2 Діаграма класів

Реалізація

У ході роботи було реалізовано патерн **Bridge** для системи експорту ментальних карт. Цей патерн дозволив розділити абстракцію (логіку експорту даних) від її реалізації (конкретного формату файлу).

Елементи патерну:

1. **MapExporter (Abstraction)** — базовий клас, що визначає високорівневу логіку експорту. Він містить посилання на об'єкт IExportFormat і делегує йому низькорівневі операції форматування.
2. **FullMapExporter (Refined Abstraction)** — конкретна абстракція, що реалізує логіку повного обходу дерева вузлів карти для експорту.
3. **IExportFormat (Implementor)** — інтерфейс, що визначає базові примітиви форматування (заголовок, вузол, підвал).
4. **TextFormat, JsonFormat (Concrete Implementors)** — класи, що реалізують специфіку запису даних у текстовому та JSON форматах відповідно.

Проблеми, які вирішує

Без використання патерну Bridge нам довелося б створювати окремі класи для кожної комбінації логіки та формату (наприклад, FullTextExporter, FullJsonExporter, FilteredTextExporter, FilteredJsonExporter). Патерн Bridge дозволяє уникнути комбінаторного вибуху класів, дозволяючи незалежно розвивати ієрархію експортерів та ієрархію форматів.

Переваги

- **Відокремлення реалізації від абстракції:** Зміна форматів файлів не впливає на логіку обходу карти.
- **Розширюваність:** Можна легко додати новий формат (наприклад, XML), просто створивши новий клас, що реалізує IExportFormat, без зміни класу MapExporter.

- **Зменшення зв'язності:** Клієнтський код (EditorForm) лише збирає "міст" (передає формат в експортер) і запускає процес.

Висновок

У ході виконання лабораторної роботи було спроєктовано та реалізовано підсистему експорту даних для системи ментальних карт "MindApp" із використанням структурного патерну **Bridge** (Міст).

Реалізовано такі ключові компоненти патерну:

- **IExportFormat** — інтерфейс реалізації (Implementor), що визначає контракт для форматування даних незалежно від структури документа.
- **TextFormat та JsonFormat** — конкретні реалізації (Concrete Implementors), які забезпечують запис даних у текстовому та JSON-форматах відповідно.
- **MapExporter** — клас абстракції (Abstraction), що керує загальним процесом експорту та делегує форматування об'єкту реалізації.
- **FullMapExporter** — уточнена абстракція (Refined Abstraction), що реалізує логіку обходу дерева вузлів.

Застосування патерну **Bridge** дозволило відокремити високорівневу логіку обходу ментальної карти від низькорівневих деталей генерації файлів. Це забезпечило гнучкість архітектури, дозволяючи додавати підтримку нових форматів без модифікації існуючого коду експортера, а також спростило структуру класів, уникнувши необхідності створення окремих підкласів для кожної комбінації формату та алгоритму експорту.

.

Код

FullMapExporter.java

client > demo > src > main > java > com > mindapp > client > patterns > FullMapExporter.java > ...

```java
package com.mindapp.client.patterns;

import com.mindapp.client.models.MindMap;
import com.mindapp.client.models.Node;

// [Refined Abstraction] - Реалізує логіку обходу всієї карти
public class FullMapExporter extends MapExporter {

    public FullMapExporter(IExportFormat format) {
        super(format);
    }

    @Override
    protected String generateBody(MindMap map) {
        if (map.getRootNode() == null) return "";
        StringBuilder sb = new StringBuilder();
        processNode(map.getRootNode(), level: 0, sb);
        return sb.toString();
    }

    private void processNode(Node node, int level, StringBuilder sb) {
        // Використовуємо Implementor для форматування конкретного вузла
        sb.append(format.formatNode(node, level));

        for (Node child : node.getChildren()) {
            processNode(child, level + 1, sb);
        }
    }
}

```

IExportFormat.java

client > demo > src > main > java > com > mindapp > client > patterns > IExportFormat.java > Language Support for Java(TM) by

```java
package com.mindapp.client.patterns;

import com.mindapp.client.models.MindMap;
import com.mindapp.client.models.Node;

// [Implementor] - Визначає базові операції форматування
public interface IExportFormat {
    String formatHeader(MindMap map);
    String formatNode(Node node, int level);
    String formatFooter(MindMap map);
    String getExtension(); // наприклад ".txt" або ".json"
}
```

JsonFormat.java

```java
package com.mindapp.client.patterns;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;
import com.mindapp.client.models.MindMap;
import com.mindapp.client.models.Node;

// [Concrete Implementor B] - Форматування у JSON (спрощено)
public class JsonFormat implements IExportFormat {
    private final ObjectMapper mapper = new ObjectMapper();

    @Override
    public String formatHeader(MindMap map) {
        return "{ \"title\": \"" + map.getTitle() + "\", \"nodes\": [\n";
    }

    @Override
    public String formatNode(Node node, int level) {
        try {
            // Створюємо простий JSON об'єкт для вузла
            ObjectNode jsonNode = mapper.createObjectNode();
            jsonNode.put("text", node.getText());
            jsonNode.put("level", level);
            return mapper.writeValueAsString(jsonNode) + ",\n";
        } catch (JsonProcessingException e) {
            return "{}";
        }
    }

    @Override
    public String formatFooter(MindMap map) {
        return "] }"; // Закриваємо масив і об'єкт
    }

    @Override
    public String getExtension() {
        return "*.json";
    }
}

```

MapExporter.java

```java
package com.mindapp.client.patterns;

import java.io.File;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;

import com.mindapp.client.models.MindMap;

// [Abstraction] - Керує процесом експорту, використовуючи Implementor
public abstract class MapExporter {
    protected IExportFormat format; // Посилання на Implementor (Bridge)

    public MapExporter(IExportFormat format) {
        this.format = format;
    }

    // Метод, який використовує формат для створення файлу
    public void export(MindMap map, File file) {
        StringBuilder content = new StringBuilder();
        content.append(format.formatHeader(map));
        content.append(generateBody(map)); // Цей крок залежить від типу експортера
        content.append(format.formatFooter(map));

        writeToFile(file, content.toString());
    }

    // Абстрактний метод, який уточнюється в RefinedAbstraction
    protected abstract String generateBody(MindMap map);

    private void writeToFile(File file, String content) {
        try {
            Files.writeString(file.toPath(), content, StandardCharsets.UTF_8);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

TextFormat.java

t > demo > src > main > java > com > mindapp > client > patterns > J TextFormat.java > Java > TextFormat

```java
package com.mindapp.client.patterns;

import com.mindapp.client.models.MindMap;
import com.mindapp.client.models.Node;

// [Concrete Implementor A] - Форматування у простий текст
public class TextFormat implements IExportFormat {
    @Override
    public String formatHeader(MindMap map) {
        return "=== MIND MAP: " + map.getTitle() + " ===\n";
    }

    @Override
    public String formatNode(Node node, int level) {
        // Відступи для ієрархії
        return "\t".repeat(level) + "- " + node.getText() + "\n";
    }

    @Override
    public String formatFooter(MindMap map) {
        return "\n(End of Export)";
    }

    @Override
    public String getExtension() {
        return "*.txt";
    }
}
```

EditorForm.java

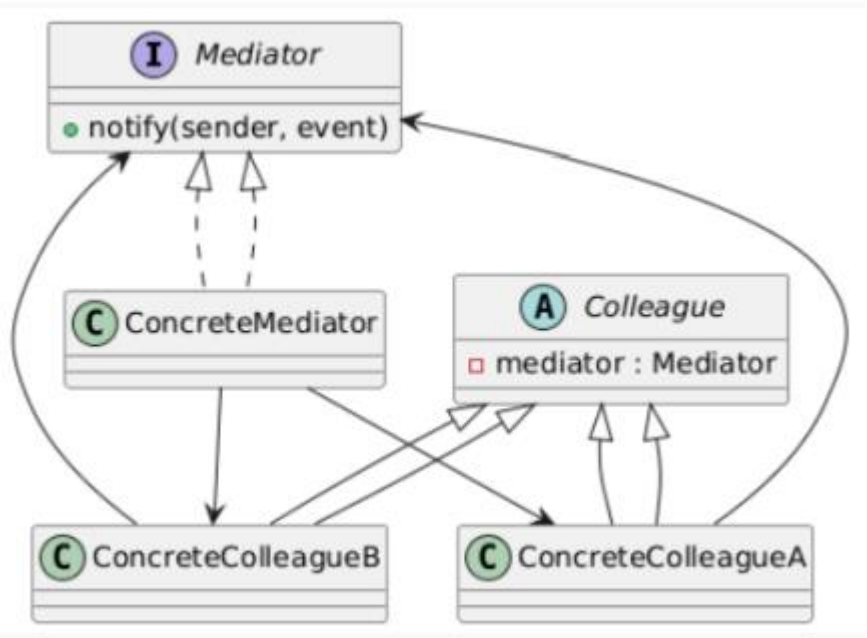
```java
private void exportMap() {
    FileChooser fileChooser = new FileChooser();
    fileChooser.setTitle(value: "Експорт мапи (Bridge Pattern)");
    fileChooser.setInitialFileName(map.getTitle());

    // Додаємо фільтри
    FileChooser.ExtensionFilter extTxt = new FileChooser.ExtensionFilter(description: "Text File", ...extensions: "*.txt");
    FileChooser.ExtensionFilter extJson = new FileChooser.ExtensionFilter(description: "JSON File", ...extensions: "*.json");
    fileChooser.getExtensionFilters().addAll(extTxt, extJson);

    File file = fileChooser.showSaveDialog(canvas.getScene().getWindow());

    if (file != null) {
        String fileName = file.getName().toLowerCase();
        IExportFormat format = null;

        // 1. Вибираємо Implementor (Формат)
        if (fileName.endsWith(suffix: ".txt")) {
            format = new TextFormat();
        } else if (fileName.endsWith(suffix: ".json")) {
            format = new JsonFormat();
        }

        if (format != null) {
            // 2. Створюємо Abstraction (Експортер) і передаємо йому Implementor
            MapExporter exporter = new FullMapExporter(format);
            exporter.export(map, file);

            new Alert(Alert.AlertType.INFORMATION, contentText: "Успішно експортовано!").show();
        }
    }
}
```

Відповіді на контрольні питання:

1. Яке призначення шаблону «Посередник»?

Шаблон Посередник (Mediator) використовується для організації взаємодії між багатьма об’єктами через центральний об’єкт-посередник, щоб зменшити кількість прямих залежностей між ними. Це знижує зв’язність та спрощує розширення логіки.

2. Нарисуйте структуру шаблону «Посередник».



3. Які класи входять в шаблон «Посередник», та яка між ними взаємодія?

- Mediator - інтерфейс посередника
- ConcreteMediator - координує об’єкти
- Colleague - базовий учасник взаємодії
- ConcreteColleague - класи, що взаємодіють через посередника

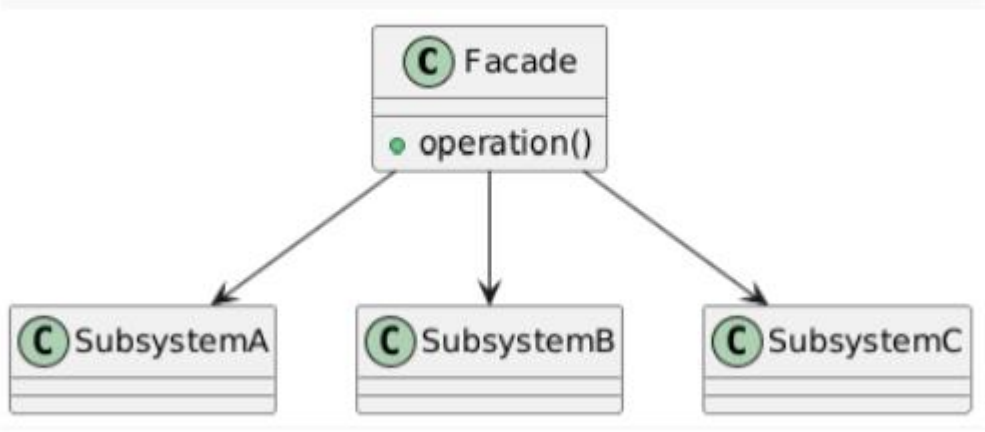
Колеги не спілкуються напряму — вони викликають методи посередника.

4. Яке призначення шаблону «Фасад»?

Шаблон Фасад (Facade) надає спрощений інтерфейс до складної підсистеми,

приховуючи внутрішні деталі реалізації й зменшуючи залежність клієнта від внутрішньої структури системи.

5. Нарисуйте структуру шаблону «Фасад»



6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?

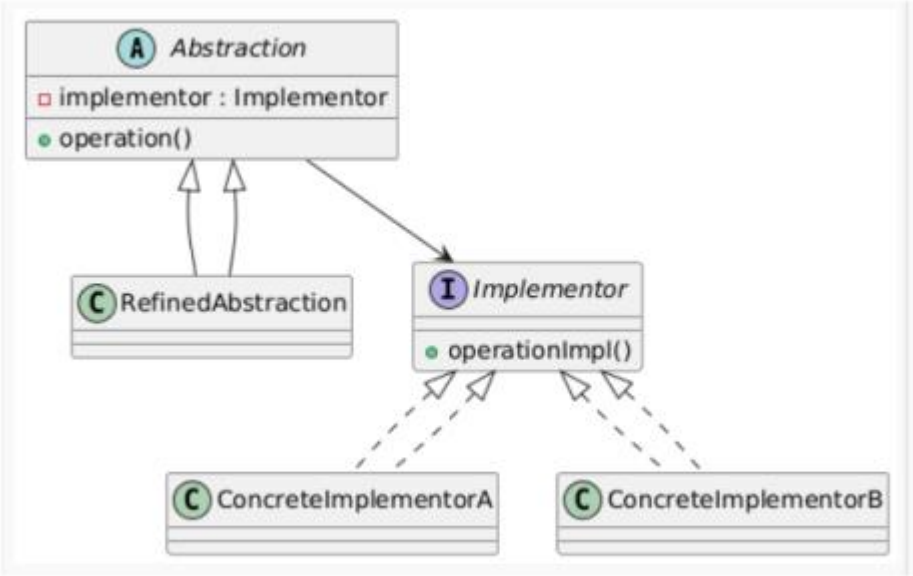
- Facade - єдина точка доступу, викликає підсистеми
- SubsystemA/B/C - внутрішні класи, з якими клієнт не працює напряму

Клієнт працює лише через фасад → зниження зв’язності.

7. Яке призначення шаблону «Міст»?

Шаблон Міст (Bridge) відділяє абстракцію від реалізації, дозволяючи їх змінювати незалежно. Замість жорсткого спадкування використовується композиція.

8. Нарисуйте структуру шаблону «Міст».



9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?

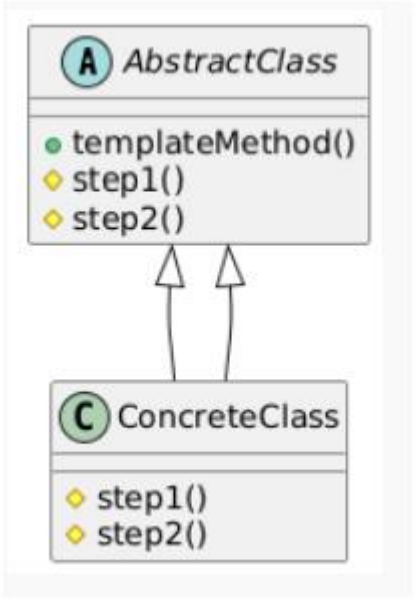
- Abstraction - високорівневий інтерфейс
- RefinedAbstraction - розширена версія абстракції
- Implementor - інтерфейс реалізації
- ConcreteImplementor - конкретні реалізації

Абстракція викликає реалізацію через композицію, а не спадкування.

10. Яке призначення шаблону «Шаблоний метод»?

Шаблон Template Method визначає скелет алгоритму у базовому класі, а підкласи можуть перевизначати окремі кроки без зміни структури алгоритму

11. Нарисуйте структуру шаблону «Шаблоний метод».



12. Які класи входять в шаблон «Шаблоний метод», та яка між ними взаємодія?

• AbstractClass - містить алгоритм та абстрактні методи

• ConcreteClass - реалізує кроки алгоритму Клієнт викликає templateMethod(), а підкласи визначають поведінку деталей.

13. Чим відрізняється шаблон «Шаблонний метод» від «Фабричного методу»?

• Шаблонний метод контролює послідовність виконання алгоритму, дозволяючи змінювати окремі етапи.

• Фабричний метод управляє створенням об'єктів, делегуючи вибір класу підкласам.

14. Яку функціональність додає шаблон «Міст»?

Шаблон Міст додає можливість розширювати абстракцію та реалізацію незалежно одна від одної, уникаючи вибуху кількості класів при комбінації різних реалізацій.

Також він:

• зменшує зв’язність між рівнями системи

• спрощує зміну реалізації під час виконання

• підтримує платформо-залежні реалізації (наприклад, різні драйвери)