



Міністерство освіти і науки України  
Національний технічний університет України  
«Київський політехнічний інститут» імені Ігоря Сікорського  
Факультет інформатики та обчислювальної техніки  
Кафедра автоматики та управління в технічних системах

## **Лабораторна робота № 2**

із дисципліни: «Технології розроблення програмного забезпечення»  
**на тему: «ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ. СЦЕНАРІЇ  
ВАРІАНТІВ ВИКОРИСТАННЯ. ДІАГРАМИ UML. ДІАГРАМИ КЛАСІВ.  
КОНЦЕПТУАЛЬНА МОДЕЛЬ СИСТЕМИ»**

Виконав:

студент групи ІА-34

Вінницький Г.Р.

Перевірив:

Мягкий Михайло Юрійович

Завдання.

1. Ознайомитися з короткими теоретичними відомостями.
2. Проаналізуйте тему та намалюйте схему прецеденту, що відповідає обраній темі лабораторії.
3. Намалюйте діаграму класів для реалізованої частини системи.
4. Виберіть 3 прецеденти і напишіть на їх основі прецеденти.
5. Розробити основні класи і структуру системи баз даних.
6. Класи даних повинні реалізувати шаблон Репозиторію для взаємодії з базою даних.
7. Підготувати звіт про хід виконання лабораторних робіт. Звіт, що подається повинен містити: діаграму прецедентів, діаграму класів системи, вихідні коди класів системи, а також зображення структури бази даних.

## **18. Shell (total commander) (state, prototype, factory method, template method, interpreter, client-server)**

Оболонка повинна вміти виконувати основні дії в системі – перегляд файлів папок в файлової системі, перемикання між дисками, копіювання, видалення, переміщення об'єктів, пошук.

## **2. Аналіз теми: Shell Total Commander**

Обрана тема присвячена розробці клієнт–серверної файлової системи Shell Total Commander, яка поєднує функціональність класичного файлового менеджера та інтерпретатора команд. Такий підхід дозволяє дослідити одразу кілька важливих аспектів сучасної програмної інженерії: роботу з файловою системою, архітектуру багаторівневих застосунків, застосування шаблонів проєктування, побудову UML-діаграм та організацію взаємодії між клієнтом і сервером.

Проєкт демонструє можливість виконання файлових операцій як локально, так і через мережеву модель. Використання TCP протоколу та JSON-серіалізації дозволяє реалізувати простий, але гнучкий канал комунікації між компонентами системи. Інтеграція SQLite як механізму логування підсилює надійність системи та додає можливість ведення історії команд.

Ключовою особливістю теми є застосування декількох шаблонів проектування — State, Factory Method, Template Method, Prototype, Interpreter, Client–Server — що забезпечує високу модульність, розширюваність та структурованість системи. Розробка графічного інтерфейсу на WPF дозволяє користувачу зручно взаємодіяти з програмою та отримувати миттєвий зворотний зв'язок у реальному часі.

Таким чином, тема є комплексною і поєднує в собі ключові елементи сучасної розробки: роботу з OS API, побудову UI, мережеву взаємодію, роботу з базою даних та застосування архітектурних рішень. Це робить її актуальною для розуміння принципів розробки системного ПЗ та практичного застосування архітектурних підходів у реальних проектах.

Схема прецеденту:

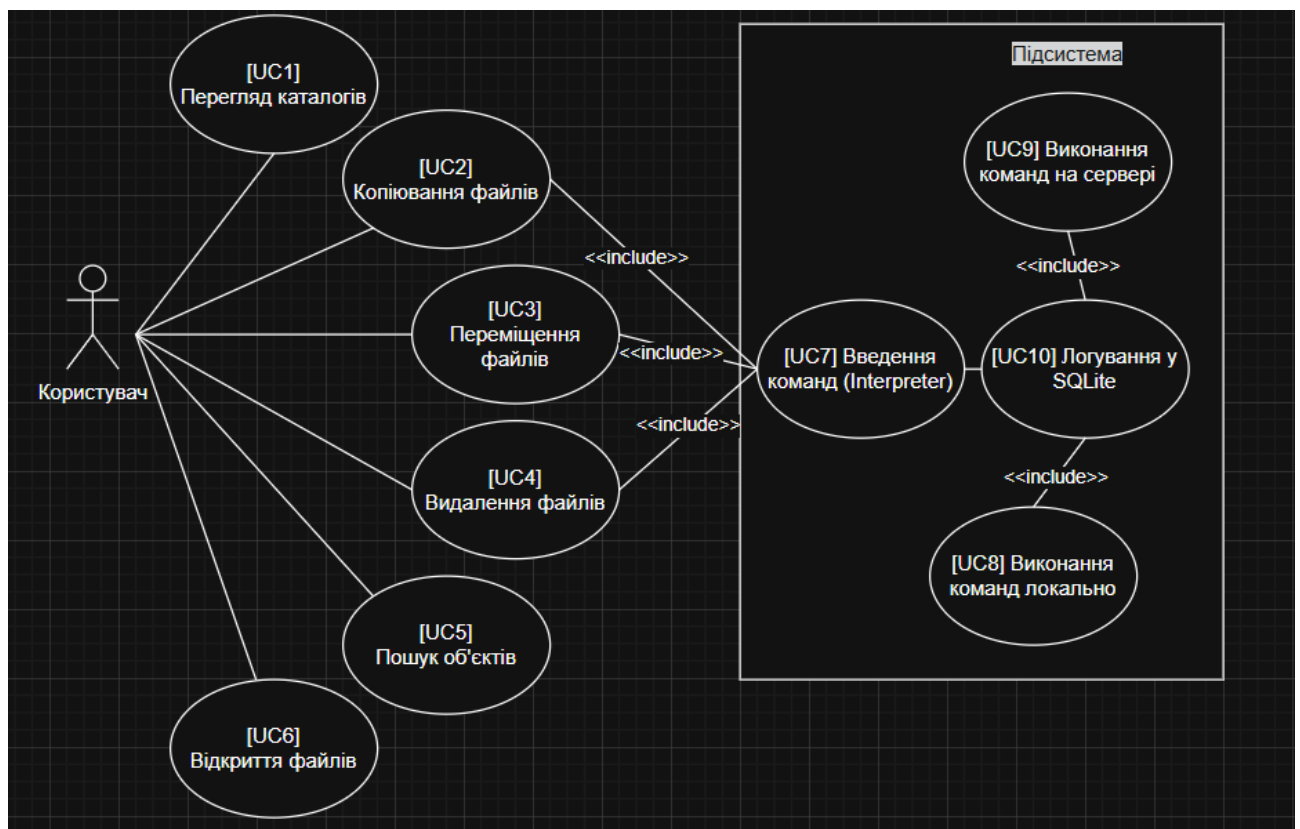


Рис1.1 – use-case діаграма функціональних вимог до системи

3.

### Діаграма класів системи:

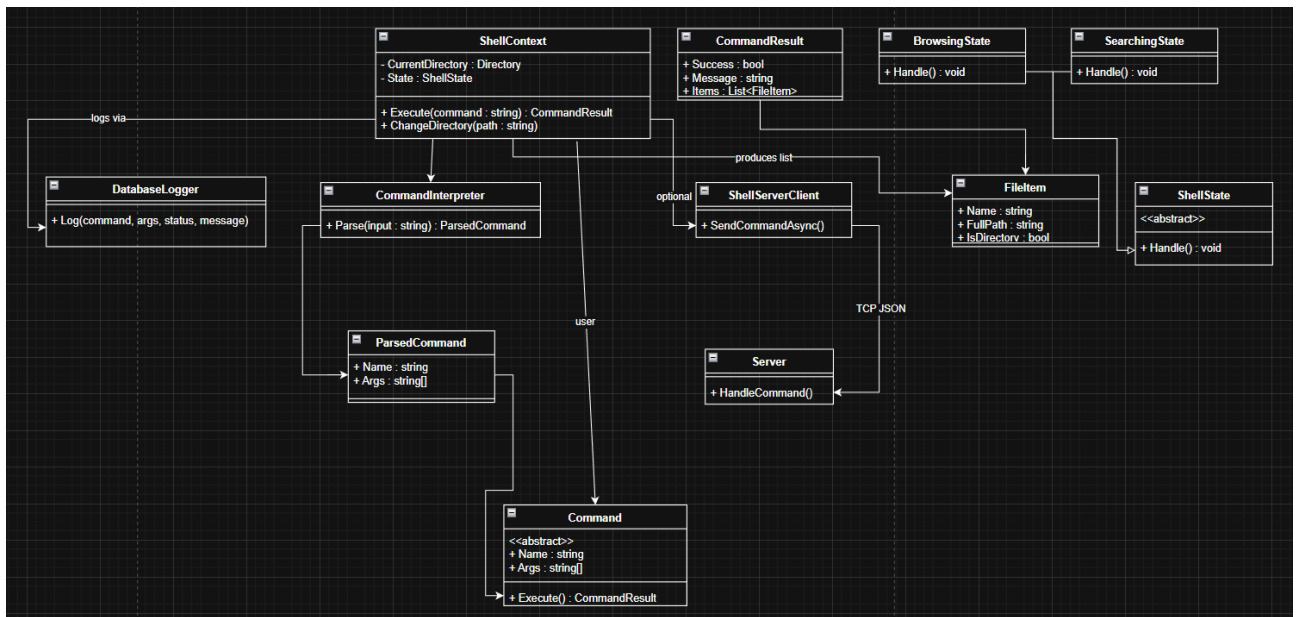


Рис. 1.2 – Діаграма класів системи

4.

### Сценарій 1. Перегляд вмісту каталогу (UC1)

Актор: Користувач

Попередні умови: Система запущена

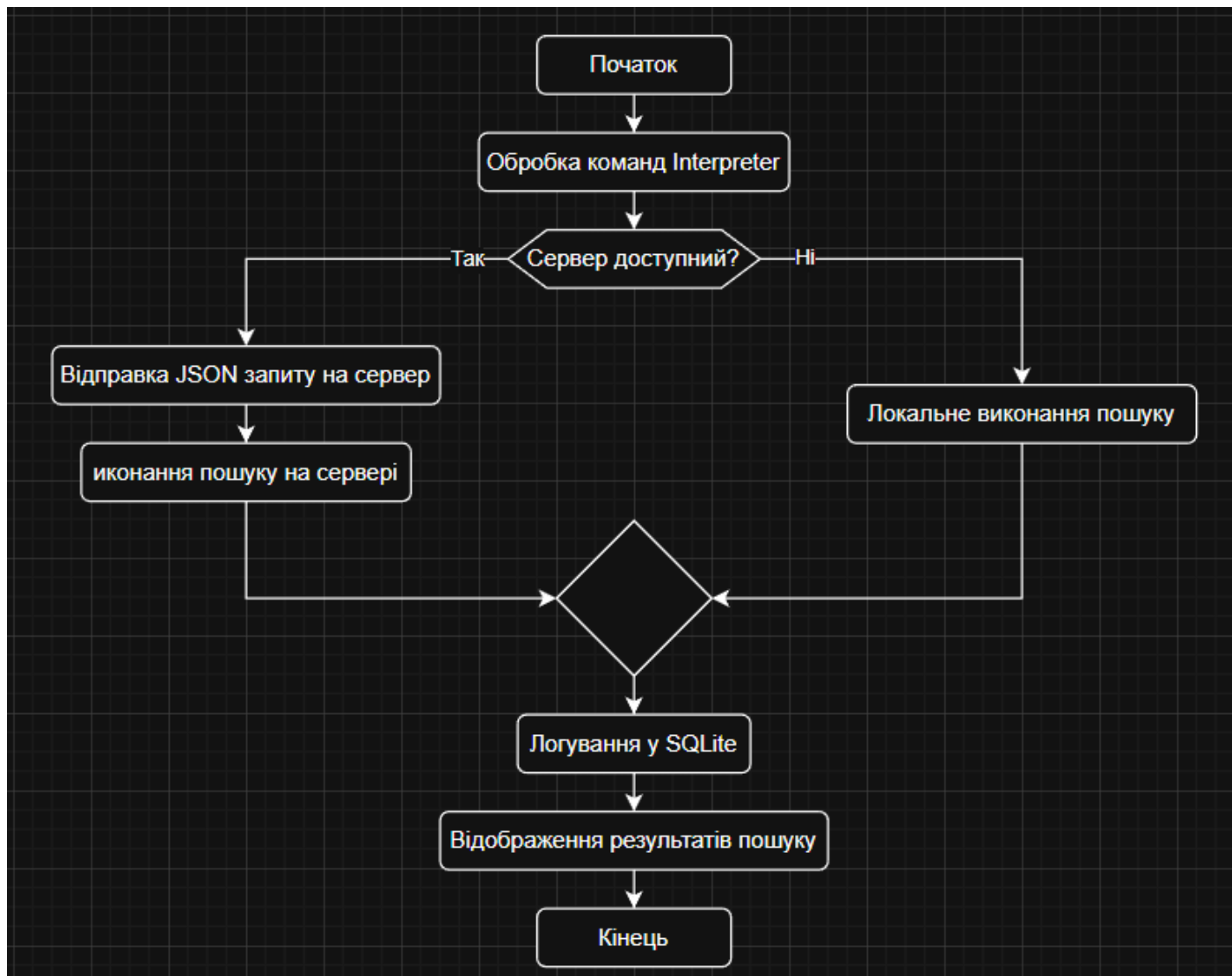
Основний сценарій:

1. Користувач відкриває програму Shell Total Commander.
2. Система автоматично відображає вміст поточної директорії.
3. Користувач може прокручувати список файлів і папок.
4. При зміні каталогу (через cd або подвійний клік) система відображає новий список файлів.

Результат: користувач бачить вміст вибраного каталогу.

### Сценарій 2. Пошук файлів у системі (UC5 + UC7 + UC8/UC9 + UC10)

Діаграма активностей:

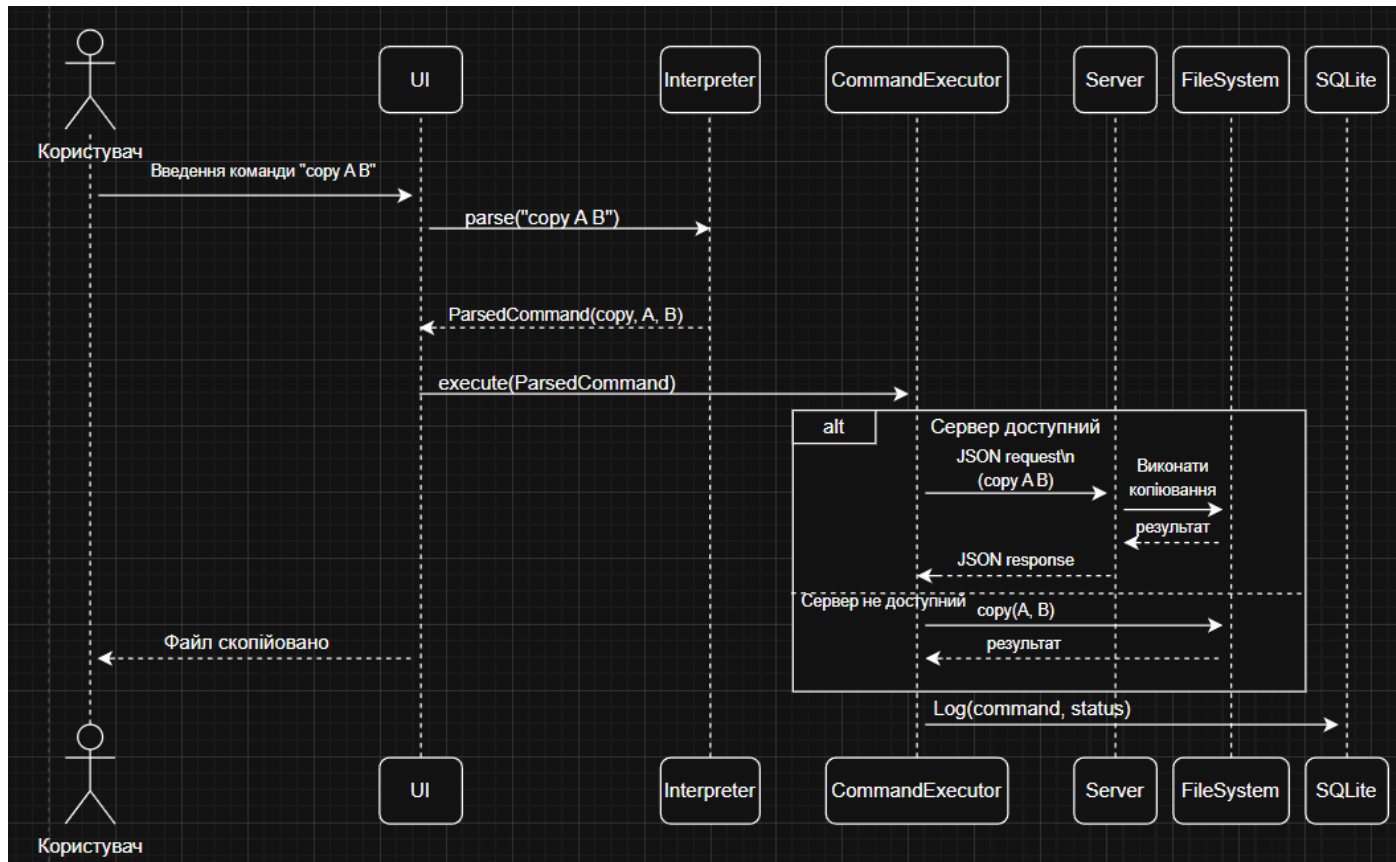


### Текстовий опис сценарію

1. Користувач вводить команду search \*.txt.
2. Інтерпретатор розпізнає команду та створює ParsedCommand.
3. Система перевіряє доступність сервера.
4. Якщо сервер доступний - пошук виконується на сервері:
  - запит пакується у JSON;
  - відправляється TCP каналом;
  - сервер шукає файли та повертає список.
5. Якщо сервер недоступний - пошук виконується локально через Directory.
6. Отримані результати логуються у SQLite.
7. UI відображає знайдені файли.

### Сценарій 3. Виконання команди копіювання через інтерпретатор (UC2 + UC7 + UC8/UC9 + UC10)

Діаграма послідовності:



Текстовий опис сценарію

1. Користувач вводить у текстовому полі команду copy A B.
2. UI передає команду інтерпретатору (Interpreter).
3. Інтерпретатор аналізує текст і формує об'єкт ParsedCommand.
4. UI передає ParsedCommand виконавцю команд (CommandExecutor у ShellContext).
5. Виконавець перевіряє:  
чи доступний сервер:  
якщо так - команда серіалізується в JSON і відправляється на TCP сервер;  
якщо ні - копіювання виконується локально на машині.
6. Після виконання результат заноситься у SQLite через DatabaseLogger.
7. UI відображає користувачеві повідомлення про успіх/помилку.
8. Вікно оновлює список файлів у поточному каталозі.

## 5. Розробити основні класи і структуру системи баз даних.

У системі використовується база даних SQLite, основне призначення якої — збереження історії виконаних команд. Для цього створюється одна таблиця:

logs(id, timestamp, command, args, status, message)

Для роботи з базою даних визначаються такі класи:

### Ключові класи:

#### 1. LogEntry

Модель даних, що представляє один запис у таблиці.

```
public class LogEntry
{
    public int Id { get; set; }
    public string Timestamp { get; set; }
    public string Command { get; set; }
    public string Args { get; set; }
    public string Status { get; set; }
    public string Message { get; set; }
}
```

#### 2. IDatabaseContext

Інтерфейс контексту бази даних.

```
public interface IDatabaseContext
{
    SqlConnection CreateConnection();
}
```

#### 3. SqliteDatabaseContext

Реалізація контексту для SQLite.

```
public class SqliteDatabaseContext : IDatabaseContext
{
    private readonly string _connectionString;

    public SqliteDatabaseContext(string dbPath)
    {
        _connectionString = $"Data Source={dbPath}";
    }

    public SqlConnection CreateConnection()
    {
        return new SqlConnection(_connectionString);
    }
}
```

## **6. Класи даних повинні реалізувати шаблон Репозиторію для взаємодії з базою даних.**

Шаблон Репозиторію забезпечує:

- 1) розділення логіки доступу до БД і бізнес-логіки;
- 2) незалежність від конкретної СУБД;
- 3) зручність тестування і розширення коду;
- 4) централізовану роботу з таблицями.

### **2.1. Інтерфейс репозиторію**

```
public interface ILogRepository
{
    Task AddAsync(LogEntry entry);
    Task<IEnumerable<LogEntry>> GetAllAsync();
    Task<LogEntry?> GetByIdAsync(int id);
}
```

### **2.2. Реалізація репозиторію SQLite**

```
public class LogRepository : ILogRepository
{
    private readonly IDatabaseContext _context;

    public LogRepository(IDatabaseContext context)
    {
        _context = context;
    }

    public async Task AddAsync(LogEntry entry)
    {
        using var conn = _context.CreateConnection();
        await conn.OpenAsync();

        var cmd = conn.CreateCommand();
        cmd.CommandText = @"
            INSERT INTO logs (timestamp, command, args, status, message)
            VALUES ($ts, $cmd, $args, $status, $msg);
        ";

        cmd.Parameters.AddWithValue("$ts", entry.Timestamp);
        cmd.Parameters.AddWithValue("$cmd", entry.Command);
        cmd.Parameters.AddWithValue("$args", entry.Args);
        cmd.Parameters.AddWithValue("$status", entry.Status);
    }
}
```

```

        cmd.Parameters.AddWithValue("$msg", entry.Message);

        await cmd.ExecuteNonQuery();
    }

    public async Task<IEnumerable<LogEntry>> GetAllAsync()
    {
        var result = new List<LogEntry>();

        using var conn = _context.CreateConnection();
        await conn.OpenAsync();

        var cmd = conn.CreateCommand();
        cmd.CommandText = "SELECT * FROM logs";

        using var reader = await cmd.ExecuteReaderAsync();
        while (await reader.ReadAsync())
        {
            result.Add(new LogEntry
            {
                Id = reader.GetInt32(0),
                Timestamp = reader.GetString(1),
                Command = reader.GetString(2),
                Args = reader.GetString(3),
                Status = reader.GetString(4),
                Message = reader.GetString(5)
            });
        }

        return result;
    }

    public async Task<LogEntry?> GetByIdAsync(int id)
    {
        using var conn = _context.CreateConnection();
        await conn.OpenAsync();

        var cmd = conn.CreateCommand();
        cmd.CommandText = "SELECT * FROM logs WHERE id=$id";
        cmd.Parameters.AddWithValue("$id", id);

        using var reader = await cmd.ExecuteReaderAsync();
        if (await reader.ReadAsync())
        {
            return new LogEntry
            {
                Id = reader.GetInt32(0),

```

```
        Timestamp = reader.GetString(1),
        Command = reader.GetString(2),
        Args = reader.GetString(3),
        Status = reader.GetString(4),
        Message = reader.GetString(5)
    };
}

return null;
}
}
```

### **Пояснення:**

**LogEntry** — модель даних, що відображає запис у таблиці logs.

**IDatabaseContext** — абстракція для відкриття з'єднання з БД.

**SqliteDatabaseContext** — під'єднання саме до SQLite.

**ILogRepository** — описує операції з таблицею.

**LogRepository** — реалізує логіку додавання та читання даних.

Репозиторій дозволяє системі:

- 1) легко змінювати тип СУБД;
- 2) отримувати дані, не знаючи SQL;
- 3) дотримуватись принципів SOLID (особливо SRP та DIP).

**Висновок:** У ході лабораторної роботи я навчився будувати різні діаграми для реалізації проекту, розробив основні класи та структуру програми.

## **Питання до лабораторної роботи:**

### **1. Що таке UML?**

UML (Unified Modeling Language) — це уніфікована мова моделювання, що використовується для опису, проєктування та документування програмних систем у вигляді графічних діаграм.

### **2. Що таке діаграма класів UML?**

Діаграма класів — це структурна діаграма UML, яка показує класи системи, їхні атрибути, методи та зв'язки між ними.

### **3. Які діаграми UML називають канонічними?**

До канонічних (основних) діаграм UML належать:

- 1) діаграма класів;
- 2) діаграма варіантів використання;
- 3) діаграма послідовностей;
- 4) діаграма станів;
- 5) діаграма діяльності;
- 6) діаграма компонентів;
- 7) діаграма розгортання.

### **4. Що таке діаграма варіантів використання?**

Це діаграма, яка показує взаємодію користувачів (акторів) із системою через варіанти використання — функції, що реалізує система.

### **5. Що таке варіант використання?**

Варіант використання (use case) — це опис певної функціональності системи, яку бачить і використовує користувач.

6. Які відношення можуть бути відображені на діаграмі використання?

На діаграмі варіантів використання використовують:

- 1) include (включення);
- 2) extend (розширення);
- 3) асоціацію актор ↔ варіант використання;
- 4) генералізацію (наслідування) акторів або use-case.

7. Що таке сценарій?

Сценарій — це конкретна послідовність кроків виконання варіанта використання, яка описує поведінку системи в певній ситуації.

8. Що таке діаграма класів?

Діаграма класів — структурна UML-діаграма, що описує логічну структуру системи: класи, їх атрибути, методи та зв'язки між ними.

9. Які зв'язки між класами ви знаєте?

Основні види зв'язків:

- 1) Асоціація
- 2) Агрегація
- 3) Композиція
- 4) Наслідування (генералізація)
- 5) Реалізація
- 6) Депенденсі (залежність)

10. Чим відрізняється композиція від агрегації?

- 1) Агрегація — слабка форма цілого-частини, частина може існувати без цілого.
- 2) Композиція — сильна форма, частина не може існувати без цілого (життєвий цикл спільний).

11. Чим відрізняється агрегація від композиції на діаграмах класів?

- 1) Агрегація позначається порожнім ромбом.
- 2) Композиція позначається зафарбованим (чорним) ромбом.
- 3) Семантика: композиція жорсткіша, означає повну залежність частини від цілого.

12. Що являють собою нормальні форми баз даних?

Нормальні форми (НФ) — це правила, що визначають рівень структурованості таблиць БД, щоб уникнути дублювання даних та аномалій оновлення.

Основні: 1НФ, 2НФ, 3НФ, Бойса–Кодда.

13. Що таке фізична модель бази даних? Логічна?

- 1) Логічна модель — структура таблиць, їх атрибутів, ключів і зв'язків, незалежна від СУБД.
- 2) Фізична модель — реальне представлення даних у конкретній СУБД: типи даних, індекси, розміщення на диску.

14. Який взаємозв'язок між таблицями БД та програмними класами?

Зв'язок зазвичай такий:

- 1) Таблиця БД  $\approx$  клас у програмі
- 2) Рядок таблиці  $\approx$  об'єкт (екземпляр класу)
- 3) Стовпець таблиці  $\approx$  поле (атрибут класу)

Це концепція ORM (Object-Relational Mapping).