Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційні систем та технологій

Лабораторна робота № 2

із дисципліни «Технології розроблення програмного забезпечення»

Тема: «Основи проектування»

Виконав
Студент групи ІА-31:
Олея М. С.

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

1. Мета: 3
2. Теоретичні відомості: 3
3. Завдання: 4
4. Хід роботи: 4
5. Висновок 12
6. Питання до лабораторної роботи: 12

1. **Мета:**

Обрати зручну систему побудови UML-діаграм та навчитися будувати діаграми варіантів використання для системи що проєктується, розробляти сценарії варіантів використання та будувати діаграми класів предметної області.

2. **Теоретичні відомості:**

Мова UML є загальноцільовою мовою візуального моделювання, яка розроблена для специфікації, візуалізації, проєктування та документування компонентів програмного забезпечення, бізнес-процесів та інших систем. Мова UML є досить строгим та потужним засобом моделювання, який може бути ефективно використаний для побудови концептуальних, логічних та графічних 18 моделей складних систем різного цільового призначення. Ця мова увібрала в себе найкращі якості та досвід методів програмної інженерії, які з успіхом використовувалися протягом останніх років при моделюванні великих та складних систем.

Діаграма – це графічне уявлення сукупності елементів моделі у формі зв'язкового графа, вершинам і ребрам (дугам) якого приписується певна семантика. Нотація канонічних діаграм є основним засобом розробки моделей мовою UML.

У нотації мови UML визначено такі види діаграм:

• варіантів використання (use case diagram);
• класів (class diagram);
• кооперації (collaboration diagram);
• послідовності (sequence diagram);
• станів (statechart diagram);
• діяльності (activity diagram);
• компонентів (component diagram);
• розгортання (deployment diagram).

Діаграма варіантів використання (Use-Cases Diagram) – це UML діаграма за допомогою якої у графічному вигляді можна зобразити вимоги до системи, що розробляється. Діаграма варіантів використання – це вихідна концептуальна модель проєктованої системи, вона не описує внутрішню побудову системи. Діаграма використання складається з:

- Акторів - будь-які об'єкти, суб'єкти чи системи, що взаємодіють з модельованою бізнес-системою ззовні для досягнення своїх цілей або вирішення певних завдань.
- Варіантів використання - служать для опису служб, які система надає актору.
- Відношень: асоціація, узагальнення, залежність, включення, розширення.

Діаграми класів використовуються при моделюванні програмних систем найчастіше. Вони є однією із форм статичного опису системи з погляду її проєктування, показуючи її структуру. Діаграма класів не відображає динамічної поведінки об'єктів зображених на ній класів. На діаграмах класів показуються класи, інтерфейси та відносини між ними.

Діаграма класів містить у собі класи, їхні методи та атрибути, зв’язки. Методи та атрибути мають 4 модифікатори доступу: public, package, protected, private. Зв’язки налічують у собі асоціацію, агрегацію, композицію, успадкування тощо.

3. Завдання:

- Ознайомитись з короткими теоретичними відомостями.
- Проаналізувати тему та спроєктувати діаграму варіантів використання відповідно до обраної теми лабораторного циклу.
- Спроєктувати діаграму класів предметної області.
- Вибрати 3 варіанти використання та написати за ними сценарії використання.
- На основі спроєктованої діаграми класів предметної області розробити основні класи та структуру бази даних системи. Класи даних повинні реалізувати шаблон Repository для взаємодії з базою даних.
- Нарисувати діаграму класів для реалізованої частини системи.
- Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму варіантів використання відповідно, діаграму класів системи, вихідні коди класів системи, а також зображення структури бази даних

Варіант - 8

Powershell terminal (strategy, command, abstract factory, bridge, interpreter,

client-server)

Термінал для powershell повинен нагадувати типовий термінал з можливістю налаштування кольорів синтаксичних конструкцій, розміру вікна, фону вікна, а також виконання команд powershell і виконуваних файлів, а також працювати в декількох вікнах терміналу (у вкладках або одночасно шляхом розділення вікна)

4. Хід роботи:

Функціональні вимоги

ID	Актор	Варіант використання	Опис
1	Користувач	Виконання команди	Введення текстової команди (PowerShell або CMD) та отримання результату її виконання.
2	Користувач	Керування вкладками	Створення нової вкладки, перемикання між ними, закриття вкладки, розділення екрану (Split view).

3	Користувач	Налаштування теми	Зміна кольору фону, шрифту, кольорів синтаксису (Strategy pattern у майбутньому).
4	Користувач	Перегляд історії	Вивід списку попередньо введених команд із можливістю їх повторного виконання.
5	Користувач	Робота з буфером	Копіювання виводу терміналу та вставка тексту з буфера обміну.
6	Система	Парсинг команди	Аналіз введеного рядка, виділення команди та аргументів (Interpreter pattern).
7	Система	Підсвітка синтаксису	Візуальне виділення ключових слів PowerShell, змінних та рядків різними кольорами.
8	Адміністратор	Запуск від імені Адміна	Запуск екземпляру терміналу з підвищеними правами доступу.

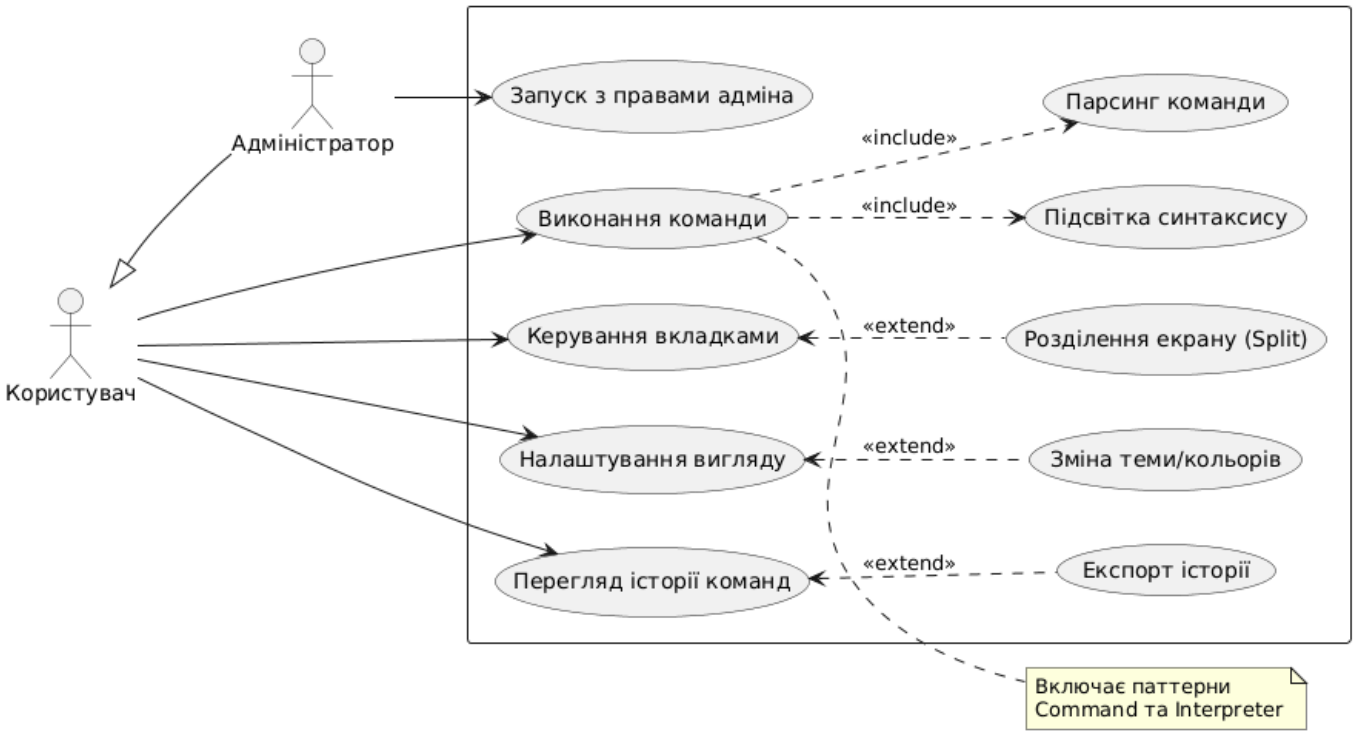

Рис. 1 - Діаграма варіантів використання

На діаграмі зображено основного актора "Користувач", який взаємодіє з системою через виконання команд, керування вікнами (вкладками) та налаштування інтерфейсу. "Виконання команди" включає в себе (include) обов'язкові процеси "Парсинг" та "Підсвітка синтаксису". Актор "Адміністратор" наслідує права Користувача, але має додаткову можливість запуску сесії з підвищеними правами (elevated privileges).

Сценарії використання:

Характеристика	Опис
Назва	Виконання PowerShell команди
Передумови	Термінал запущено, курсор знаходиться в рядку вводу.
Постумови	Команда виконана, результат виведено на екран, команду збережено в історії.
Взаємодіючі сторони	Користувач, Інтерпретатор команд.
Основний потік подій	1. Користувач вводить команду (наприклад, Get-Process). 2. Користувач натискає Enter. 3. Система парсить текст (Interpreter). 4. Система створює об'єкт команди (Command). 5. Система виконує команду і повертає текст результату. 6. Система відображає результат.
Винятки	Команду не знайдено або помилка синтаксису. Система виводить повідомлення про помилку червоним кольором.

Табл. 1 Виконання команди

Характеристика	Опис
Назва	Зміна колірної схеми терміналу
Передумови	Термінал запущено, відкрито меню налаштувань.
Постумови	Інтерфейс терміналу змінив кольори фону та тексту. Конфігурація збережена в БД/файлі.
Взаємодіючі сторони	Користувач, Менеджер конфігурацій.
Основний потік подій	1. Користувач обирає опцію "Налаштування". 2. Користувач обирає нову тему зі списку (наприклад, "Dark Matrix"). 3. Система застосовує параметри теми (колір фону, шрифт) до активного вікна. 4. Система зберігає вибір у профіль користувача.
Винятки	Файл конфігурації пошкоджено. Завантажується тема за замовчуванням.

Табл. 2 Налаштування теми

Характеристика	Опис
Назва	Відкриття нової вкладки
Передумови	Запущено хоча б одне вікно терміналу.
Постумови	Створено нову незалежну сесію PowerShell в новій вкладці.
Взаємодіючі сторони	Користувач, Менеджер вікон.
Основний потік	1. Користувач натискає кнопку "+" або комбінацію клавіш (Ctrl+T). 2. Система ініціалізує нову сесію (Abstract Factory). 3. Система додає вкладку в панель вкладок.

	4. Фокус перемикається на нову вкладку.
Винятки	Досягнуто ліміту пам'яті. Система видає попередження.
Характеристика	Опис

Табл. 3 Робота з вкладками

Структура Бази Даних

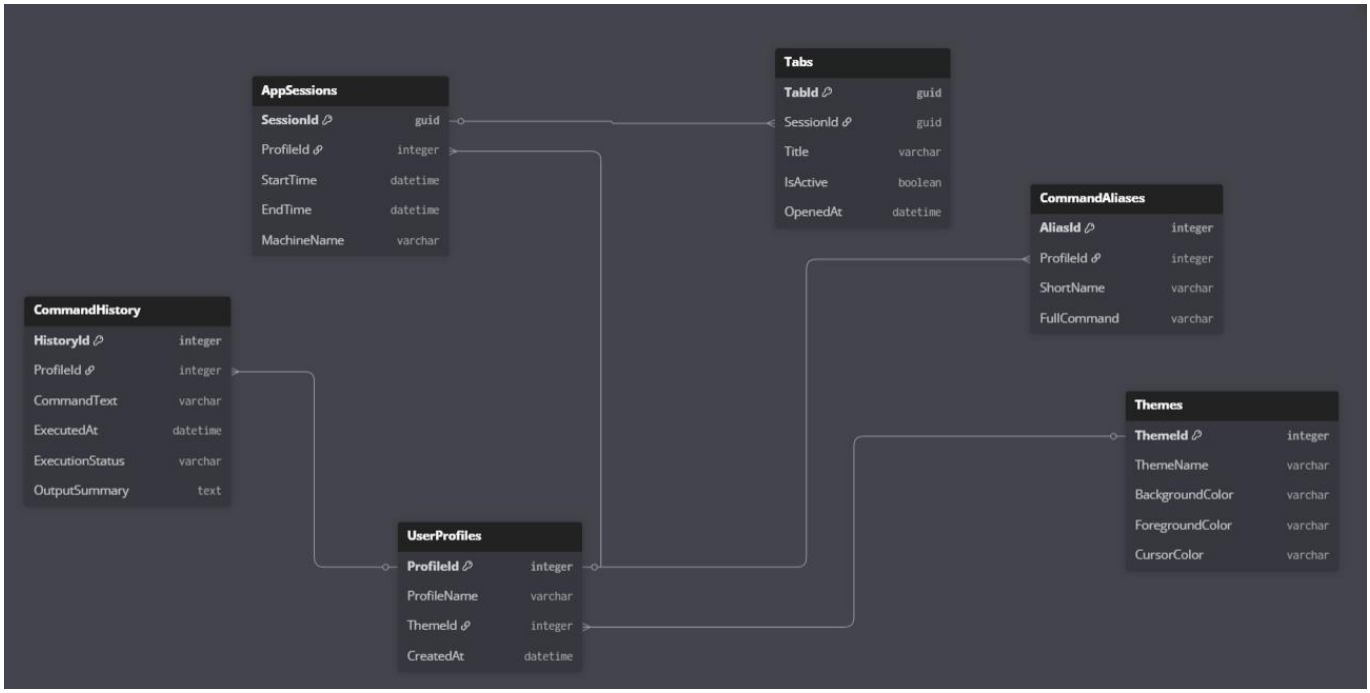

Рис. 2 - Структура БД

UserProfiles (Профілі налаштувань): Центральна сутність, що зберігає конфігурацію користувача. Дозволяє мати кілька пресетів налаштувань (наприклад, окремо для роботи та дому). Зв'язок: Має зовнішній ключ ThemeId для зв'язку з візуальною темою.

Themes (Візуальні теми): Довідкова таблиця, що містить параметри кольорової схеми (фон, текст, курсор). Зв'язок: Використовується в UserProfiles (зв'язок Один-до-Багатьох: одна тема може бути у багатьох профілів).

AppSessions (Сесії роботи): Журнал запусків термінала. Фіксує час початку та завершення роботи програми. Зв'язок: Посилається на UserProfiles, щоб знати, з якими налаштуваннями було запущено термінал.

Tabs (Вкладки термінала): Відображає динамічну структуру інтерфейсу. Одна сесія може містити багато відкритих вкладок. Зв'язок: Посилається на AppSessions.

CommandHistory (Журнал команд): Зберігає введені користувачем команди для реалізації функції "історія" (стрілка вгору/вниз) та аудиту. Зв'язок: Прив'язана до UserProfiles, що дозволяє зберігати історію між різними сесіями (перезапусками

програми). CommandAliases (Користувацькі скорочення): Дозволяє користувачеві створювати власні короткі команди (аліаси) для довгих скриптів. Зв'язок: Персоналізовані для кожного UserProfiles.

Діаграми класів:

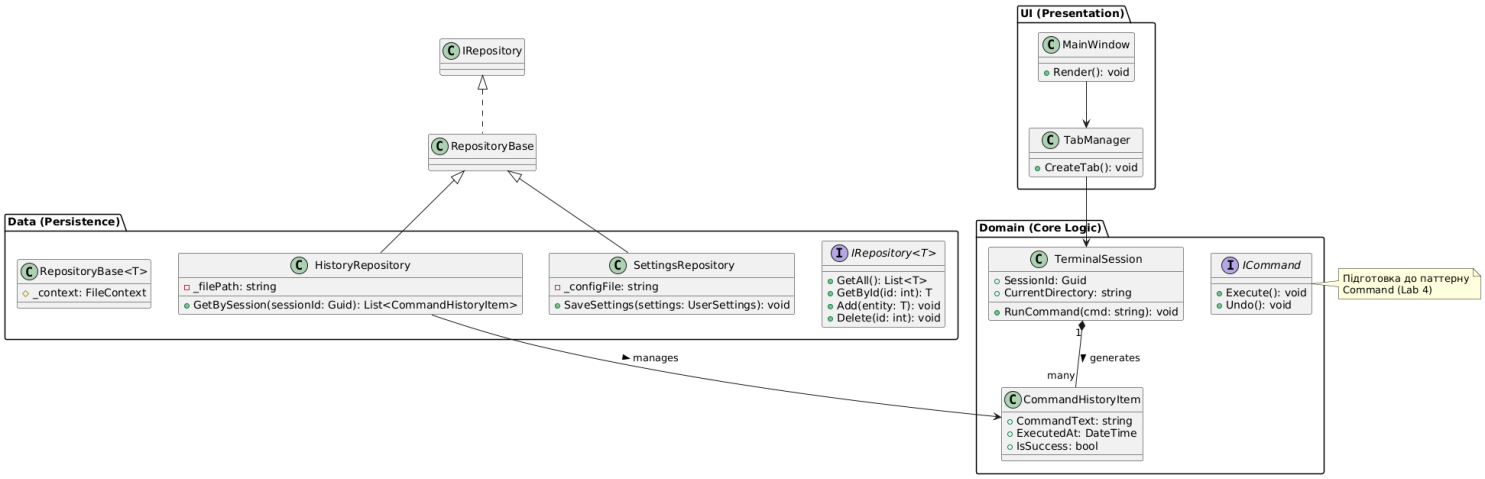

Рис. 3 - Діаграма класів

Клас / Інтерфейс	Тип	Опис та призначення
ICommand	Interface	Базовий інтерфейс для реалізації патерну Command. Визначає контракт для всіх операцій, що виконуються в терміналі. Метод Execute() запускає логіку команди, а Undo() дозволяє скасувати дію (наприклад, видалення файлу).
TerminalSession	Class	Центральний клас логіки. Відповідає за один активний сеанс роботи (одну вкладку). Зберігає поточний стан (наприклад, CurrentDirectory — робочу папку) та метод RunCommand(), який передає введену користувачем команду на обробку інтерпретатору.
CommandHistoryItem	Class	Клас-сутність (DTO), що представляє один запис в історії команд. Зберігає текст команди (CommandText), час виконання (ExecutedAt) та статус успішності (IsSuccess). Використовується для формування логів.
IRepository<T>	Interface	Узагальнений (Generic) інтерфейс, що описує стандартні CRUD-операції (GetAll, GetById, Add, Delete). Дозволяє уніфікувати роботу з різними типами даних.
RepositoryBase<T>	Abstract Class	Базова реалізація репозиторію. Містить спільну логіку підключення до контексту даних (FileContext), щоб уникнути дублювання коду в конкретних репозиторіях.
HistoryRepository	Class	Специфічний репозиторій для роботи з історією команд. Реалізує метод GetBySession(), що дозволяє завантажити

		історію для конкретної вкладки або сесії користувача. Працює з локальним файловим сховищем.
SettingsRepository	Class	Відповідає за збереження та завантаження конфігурації користувача (теми, шрифти). Метод SaveSettings() серіалізує об'єкт налаштувань у конфігураційний файл.
MainWindow	Class	Головне вікно програми. Відповідає за ініціалізацію компонентів інтерфейсу та метод Render(), який оновлює зображення на екрані.
TabManager	Class	Менеджер вкладок. Відповідає за створення (CreateTab), закриття та перемикання між вкладками термінала. Він пов'язує візуальну вкладку з логічним об'єктом TerminalSession.

Табл. 4 Опис класів спроєктованої системи

Опис зв'язків (Relationships)

На діаграмі відображено ключові типи взаємодії між класами:

1. Реалізація (Realization): Клас RepositoryBase реалізує інтерфейс IRepository, забезпечуючи дотримання контракту.
2. Успадкування (Inheritance): HistoryRepository та SettingsRepository успадковують функціонал від RepositoryBase, розширюючи його специфічними методами.
3. Композиція (Composition): TerminalSession володіє списком об'єктів CommandHistoryItem. Це означає, що історія команд нерозривно пов'язана з сесією (в контексті роботи пам'яті).
4. Асоціація (Association): MainWindow використовує TabManager для керування вкладками, а TabManager створює екземпляри TerminalSession.

Програмна реалізація:

a) UserProfile.cs (Сутність профілю)

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
namespace PowerShellTerminal.App.Domain.Entities
{
    public class UserProfile
    {
        [Key]
        public int ProfileId { get; set; }
        public string ProfileName { get; set; } = string.Empty;
```

```
        public int ThemeId { get; set; }
        public DateTime CreatedAt { get; set; }
        public Theme? Theme { get; set; }
        public ICollection<CommandHistoryItem> HistoryItems { get; set; } = new
List<CommandHistoryItem>();
    }
```

б) Theme.cs (Сутність теми)

```
using System.ComponentModel.DataAnnotations;

namespace PowerShellTerminal.App.Domain.Entities
{
    public class Theme
    {
        [Key]
        public int ThemeId { get; set; }
        public string ThemeName { get; set; } = string.Empty;
        public string BackgroundColor { get; set; } = "#000000";
        public string ForegroundColor { get; set; } = "#FFFFFF";
        public string CursorColor { get; set; } = "#FFFFFF";
    }
}
```

в) CommandHistoryItem.cs (Сутність історії)

```
using System;
using System.ComponentModel.DataAnnotations;

namespace PowerShellTerminal.App.Domain.Entities
{
    public class CommandHistoryItem
    {
        [Key]
        public int HistoryId { get; set; }

        public int ProfileId { get; set; }

        public string CommandText { get; set; } = string.Empty;
        public DateTime ExecutedAt { get; set; }
        public bool IsSuccess { get; set; }
        public UserProfile? Profile { get; set; }
    }
}
```

г) TerminalSession.cs (Логіка сесії)

```
using System;
using System.Collections.Generic;

namespace PowerShellTerminal.App.Domain.Entities
{
```

```
    public class TerminalSession
    {
        public Guid SessionId { get; set; }
        public string CurrentDirectory { get; set; }
        public bool IsActive { get; set; }

        public TerminalSession()
        {
            SessionId = Guid.NewGuid();
            CurrentDirectory = Environment.CurrentDirectory;
            IsActive = true;
        }

        public void RunCommand(string command)
        {
            Console.WriteLine($"Executing: {command}...");
        }
    }
}
```

е) IRepository.cs (Інтерфейс)

```
using System.Collections.Generic;

namespace PowerShellTerminal.App.Domain.Interfaces
{
    public interface IRepository<T> where T : class
    {
        IEnumerable<T> GetAll();
        T? GetById(int id);
        void Add(T entity);
        void Delete(int id);
    }
}
```

є) HistoryRepository.cs (Реалізація репозиторію)

```
using System.Collections.Generic;
using PowerShellTerminal.App.Domain.Entities;
using PowerShellTerminal.App.Domain.Interfaces;

namespace PowerShellTerminal.App.Data
{
    public class HistoryRepository : IRepository<CommandHistoryItem>
    {
        private readonly List<CommandHistoryItem> _memoryDb = new
List<CommandHistoryItem>();

        public void Add(CommandHistoryItem entity)
        {
            _memoryDb.Add(entity);
        }
```

```
        public void Delete(int id)
        {
            var item = GetById(id);
            if (item != null)
            {
                _memoryDb.Remove(item);
            }
        }

        public IEnumerable<CommandHistoryItem> GetAll()
        {
            return _memoryDb;
        }

        public CommandHistoryItem? GetById(int id)
        {
            return _memoryDb.Find(x => x.HistoryId == id);
        }
    }
}
```

5. Висновок

У ході виконання лабораторної роботи було здійснено системний аналіз предметної області та спроєктовано архітектуру програмного засобу «PowerShell Terminal». Розроблено комплекс UML-моделей, зокрема діаграми варіантів використання та класів, що дозволило чітко визначити функціональні вимоги та логіку взаємодії компонентів системи. Сформовано нормалізовану структуру даних для збереження налаштувань профілів і історії команд, а також реалізовано програмний каркас проєкту мовою C# із застосуванням патерну Repository для забезпечення абстракції рівня доступу до даних. Отримані результати створюють надійну основу для подальшої імплементації шаблонів проєктування Command, Interpreter та Strategy у наступних етапах розробки.

6. Питання до лабораторної роботи:

1) Що таке UML? UML (Unified Modeling Language) — уніфікована мова моделювання, стандартна графічна нотація для візуалізації, специфікації, конструювання та документування систем, переважно програмних.
2) Що таке діаграма класів UML? Діаграма класів UML — структурна діаграма, що відображає класи системи, їх атрибути, операції та відношення між ними.
3) Які діаграми UML називають канонічними? Канонічними (основними) діаграмами UML часто називають діаграму варіантів використання (use

case diagram), діаграму класів (class diagram), діаграму послідовності (sequence diagram) та діаграму діяльності (activity diagram), як найбільш поширені та фундаментальні.

4) Що таке діаграма варіантів використання? Діаграма варіантів використання (use case diagram) — поведінкова діаграма UML, що показує функціональність системи з точки зору зовнішніх користувачів (акторів), їх взаємодію з системою через варіанти використання.
5) Що таке варіант використання? Варіант використання (use case) — опис послідовності дій, які система виконує для досягнення певної мети для актора (користувача).
6) Які відношення можуть бути відображені на діаграмі використання? На діаграмі варіантів використання відображаються: асоціація (між актором і use case), узагальнення (генералізація акторів або use case), включення (<<include>>), розширення (<<extend>>).
7) Що таке сценарій? Сценарій — конкретна послідовність дій та взаємодій у варіанті використання, що описує один можливий шлях виконання (основний або альтернативний).
8) Що таке діаграма класів? Діаграма класів — статична структурна діаграма UML, що моделює структуру системи через класи, інтерфейси, їх атрибути, методи та зв'язки між ними.
9) Які зв’язки між класами ви знаєте? Основні зв’язки: асоціація, агрегація, композиція, узагальнення (спадкування), залежність, реалізація (інтерфейсу).
10) Чим відрізняється композиція від агрегації? Композиція — сильніший зв'язок "ціле-частина", де частини не можуть існувати без цілого (знищення цілого знищує частини). Агрегація — слабший зв'язок, де частини можуть існувати незалежно від цілого.
11) Чим відрізняється зв'язки типу агрегації від зв'язків композиції на діаграмах класів? На діаграмах класів агрегація позначається порожнім ромбом на стороні цілого, композиція — заповненим ромбом. Композиція означає сильнішу залежність життєвого циклу частин від цілого.
12) Що являють собою нормальні форми баз даних? Нормальні форми — правила організації даних у реляційних БД для усунення надмірності та аномалій (вставки, видалення, оновлення). Основні: 1НФ, 2НФ, 3НФ, НФБК, 4НФ, 5НФ.
13) Що таке фізична модель бази даних? Логічна? Логічна модель — опис структури даних (таблиці, стовпці, ключі, зв'язки) незалежно від конкретної СУБД. Фізична модель — детальний опис реалізації (індекси, партиції, типи зберігання, файли) для конкретної СУБД.
14) Який взаємозв'язок між таблицями БД та програмними класами? У об'єктно-реляційному відображенні (ORM) таблиці БД часто відповідають класам (рядки — об'єктам, стовпці — атрибутам), а зв'язки

між таблицями (foreign key) — асоціаціям, агрегаціям чи композиціям між класами.