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

**Лабораторна робота № 2**

з дисципліни «Теорія розробки прогрманого забезпечення»

Тема: «Основи проектування.»

Варіант «Powershell terminal»

Виконав:

студент групи ІА-32
Бечке Олексій Ігорович

Перевірив:

Мягкий Михайло Юрійович

Київ 2025

## Зміст



## Вступ

**Тема:** Основи проектування.

**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.

Проєктування інформаційних систем є одним з ключових етапів їх розробки, адже від правильності побудови моделей залежить якість та ефективність кінцевого продукту. UML (Unified Modeling Language) є універсальною мовою моделювання, що дозволяє формалізувати структуру та поведінку системи, роблячи її зрозумілою як для розробників, так і для користувачів.

## 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 (diagram)** – 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.

**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 варіантів використання складається з кількох основних елементів: прецедентів , акторів та відносин між ними.

**Сценарії використання** – це текстові уявлення тих процесів, які відбуваються при взаємодії користувачів системи та самої системи. Вони є чітко формалізованими, покроковими інструкціями, що описують той чи інший процес у термінах кроків досягнення мети. Сценарії використання однозначно визначають кінцевий результат.

**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.

**Клас** – це основний будівельний блок програмної системи. Це поняття є і в мовах програмування, тобто між класами UML та програмними класами є відповідність, що є основою для автоматичної генерації програмних кодів або для виконання реінжинірингу. Кожен клас має назву, атрибути та операції. Клас на діаграмі показується як прямокутник, розділений на 3 області. У верхній міститься назва класу, у середній – опис атрибутів (властивостей), у нижній – назви операцій – послуг, що надаються об'єктами цього класу.

**Логічна модель бази даних** є структурою таблиць, уявлень, індексів та інших логічних елементів бази даних, що дозволяють власне програмування та використання бази даних.

Процес створення логічної моделі бази даних зветься проєктування бази даних (database design). Проєктування відбувається у зв'язку з опрацюванням архітектури програмної системи, оскільки база даних створюється зберігання даних, одержуваних з програмних класів.

## Хід роботи

### 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.

**Діаграма варіантів використання**

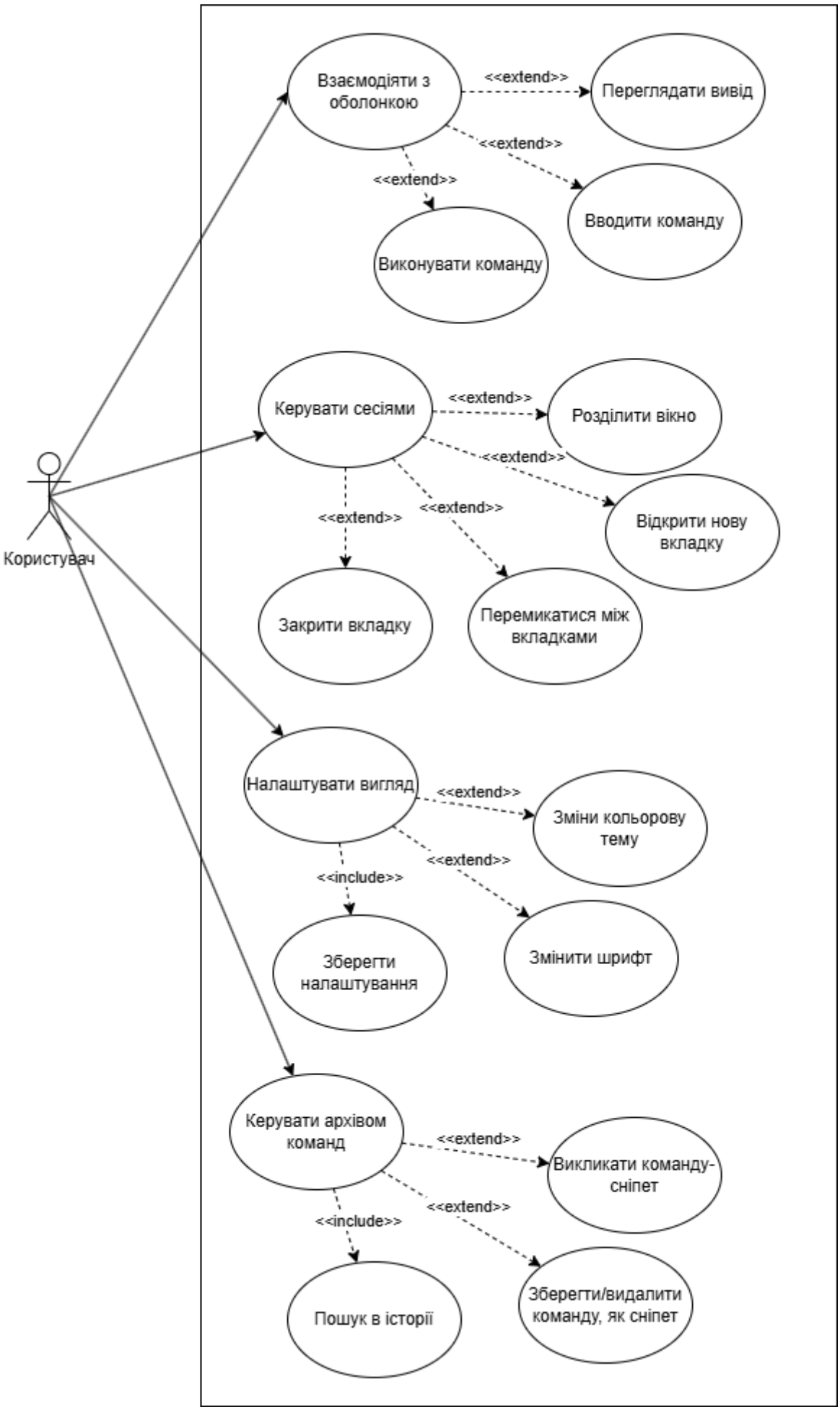

Користувач
Взаємодіяти з оболонкою
<<extend>>
Переглядати вивід
<<extend>>
Вводити команду
<<extend>>
Виконувати команду
Керувати сесіями
<<extend>>
Розділити вікно
<<extend>>
Відкрити нову вкладку
<<extend>>
Закрити вкладку
<<extend>>
Перемикатися між вкладками
Налаштувати вигляд
<<extend>>
Зміни кольорову тему
<<include>>
Зберегти налаштування
<<extend>>
Змінити шрифт
Керувати архівом команд
<<extend>>
Викликати команду-сніпет
<<include>>
Пошук в історії
<<extend>>
Зберегти/видалити команду, як сніпет

Рис. 1 – Діаграма використання (Use-Case diagram)

**3 сценарії використання**

Оберемо 3 ключових варіанти використання та опишемо їх.

**Сценарій 1:** Виконання команди PowerShell

- Передумови:
  - Додаток "PowerShell Terminal" запущено.
  - Існує принаймні одна активна сесія (вкладка або панель).
  - Фокус вводу знаходиться у текстовій області активної сесії.
- Постумови:
  - Введена користувачем команда виконана процесом powershell.exe.
  - Результат виконання (стандартний вивід або потік помилок) відображений у текстовій області терміналу.
  - Сесія готова до введення наступної команди.
- Взаємодіючі сторони:
  - Користувач (ініціатор).
- Короткий опис:
  - Сценарій описує основну функцію терміналу: введення користувачем текстової команди PowerShell, її виконання та відображення результату.
- Основний перебіг подій:
  1. Користувач вводить текстову команду з клавіатури (напр., Get-ChildItem -Path C:\).
  2. Система відображає символи, що вводяться, в активній текстовій області.
  3. Користувач натискає клавішу Enter для підтвердження виконання.
  4. Система отримує введений текстовий рядок.
  5. Система надсилає рядок команди у стандартний потік вводу (stdin) відповідного процесу powershell.exe.
  6. Система очікує та зчитує дані зі стандартного потоку виводу (stdout) процесу powershell.exe.
  7. Система відображає отримані дані у текстовій області, додаючи їх після введеної команди.
  8. Система повертається в режим очікування вводу від користувача.
- Винятки:
  - Введено порожню команду: Якщо користувач натискає Enter без введення тексту, система ігнорує дію і повертається до кроку 8.
  - Команда не знайдена або виконана з помилкою: Система зчитує дані зі стандартного потоку помилок (stderr) процесу powershell.exe. Система відображає текст помилки
- Примітки:

- Кожна команда та її вивід додаються до історії сесії, доступної для перегляду шляхом прокручування.

**Сценарій 2:** Відкриття нової вкладки

- Передумови:
  - Додаток "PowerShell Terminal" запущено.
- Постумови:
  - Створено нову, незалежну сесію PowerShell.
  - У графічному інтерфейсі з'явилася нова вкладка, що відповідає новій сесії.
  - Нова вкладка є активною, і користувач може вводити в ній команди.
- Взаємодіючі сторони:
  - Користувач (ініціатор).
- Короткий опис:
  - Сценарій описує процес створення нової ізольованої сесії PowerShell, представленої у вигляді вкладки в головному вікні програми.
- Основний перебіг подій:
  1. Користувач натискає на елемент керування "Нова вкладка" (зазвичай кнопка +).
  2. Система створює новий об'єкт TerminalSession.
  3. Система запускає новий процес powershell.exe в операційній системі.
  4. Система пов'язує потік вводу/виводу нового процесу з об'єктом TerminalSession.
  5. Система додає новий віджет вкладки до TabPane у головному вікні.
  6. Система робить новостворену вкладку активною (перемикає фокус на неї).
  7. Система відображає початкове запрошення до вводу від PowerShell у текстовій області нової вкладки.
- Винятки:
  - Не вдалося запустити процес: Операційна система не може запустити powershell.exe (напр., файл не знайдено або недостатньо прав). Система відображає повідомлення про помилку у новій вкладці або в окремому діалоговому вікні.
- Примітки:
  - Нова сесія не успадковує стан (поточну директорію, змінні) від інших відкритих сесій.

**Сценарій 3:** Зміна кольорової схеми

- Передумови:
  - Додаток "PowerShell Terminal" запущено.
- Постумови:
  - Нові налаштування кольорів застосовані до всіх відкритих та майбутніх сесій.
  - Зміни збережені у сховищі даних (БД або файл конфігурації) для використання при наступних запусках програми.
- Взаємодіючі сторони:
  - Користувач (ініціатор).
- Короткий опис:
  - Сценарій описує, як користувач змінює налаштування кольорів для елементів інтерфейсу та синтаксису, і як ці зміни зберігаються та застосовуються.
- Основний перебіг подій:
  1. Користувач відкриває вікно налаштувань через відповідний пункт меню.
  2. Система завантажує поточні налаштування зі сховища (через ThemeRepository) і відображає їх у вікні налаштувань.
  3. Користувач переходить до вкладки "Вигляд" або "Кольори".
  4. Користувач вибирає елемент для налаштування (напр., "Фон", "Ключове слово").
  5. Користувач обирає новий колір за допомогою палітри кольорів.
  6. Система оновлює попередній перегляд у вікні налаштувань.
  7. Користувач натискає кнопку "Зберегти" або "Застосувати".
  8. Система зберігає новий об'єкт Theme у сховищі даних.
  9. Система оновлює вигляд усіх відкритих вкладок відповідно до нових налаштувань.
  10. Користувач закриває вікно налаштувань.
- Винятки:
  - Користувач натискає "Скасувати": Система закриває вікно налаштувань, ігноруючи будь-які зроблені зміни.
  - Помилка збереження: Системі не вдалося зберегти налаштування (напр., немає доступу до файлу бази даних). Система відображає повідомлення про помилку.
- Примітки:
  - Зміни можуть застосовуватися "на льоту" (одразу після вибору кольору) для кращого досвіду користувача. В такому випадку кнопка "Застосувати" може бути відсутня.

## Діаграма класів

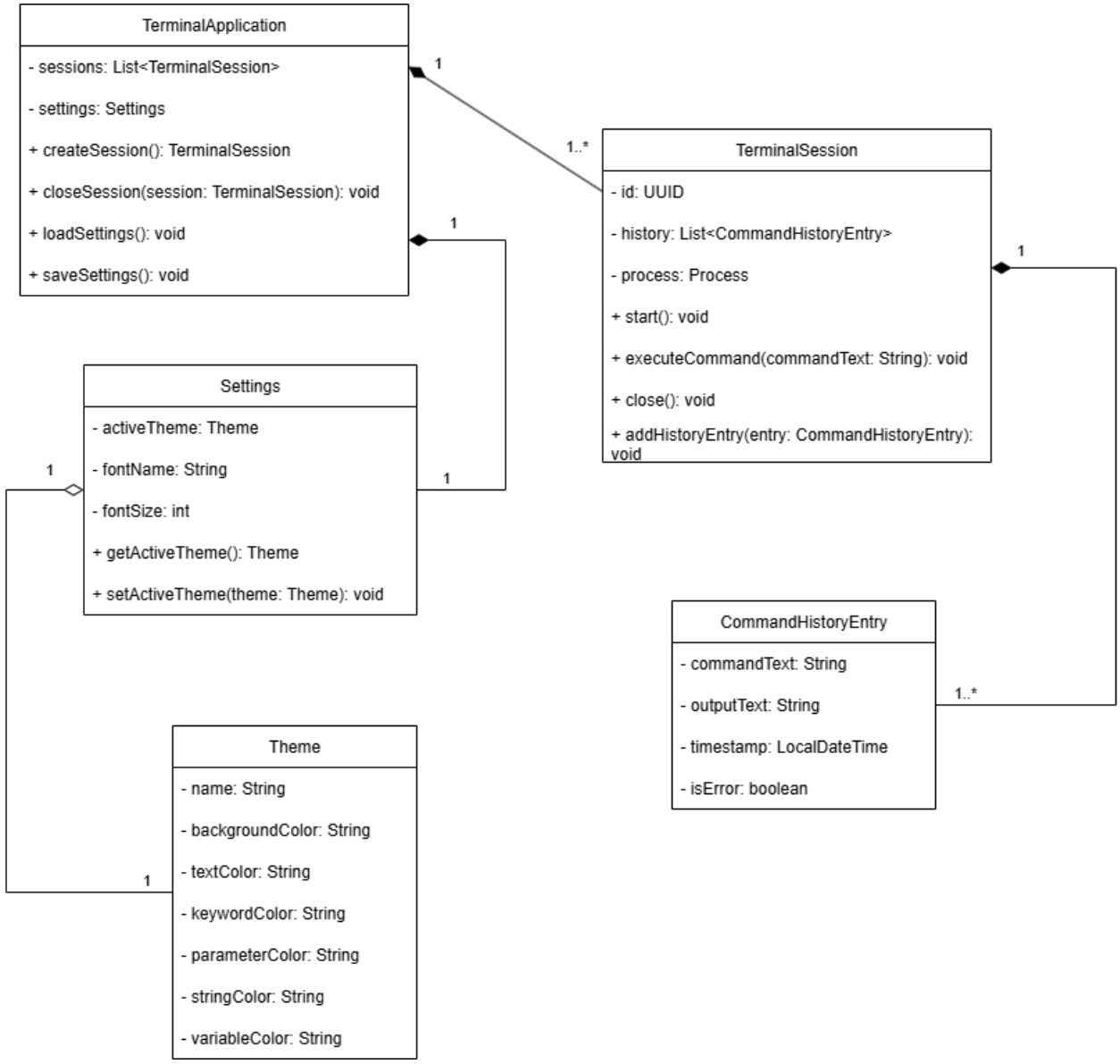



Рис. 2 – діаграма класів (Class diagram)

**TerminalApplication – TerminalSession:**

Композиція. TerminalApplication створює та володіє об'єктами TerminalSession.

**TerminalApplication – Settings:**

Композиція. TerminalApplication володіє єдиним екземпляром Settings.

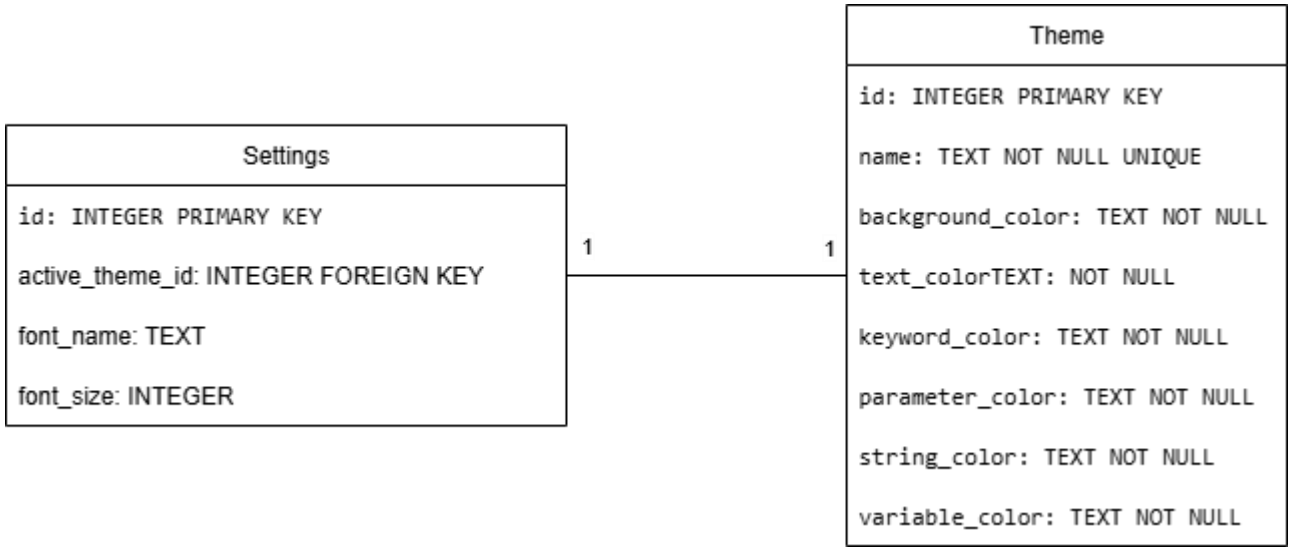
**TerminalSession – CommandHistoryEntry:**

Композиція. TerminalSession створює та володіє записами своєї історії.

**Settings – Theme:**

Агрегація. Settings має посилання на об'єкт Theme, але Theme може існувати і як самостійна сутність (наприклад, у списку доступних тем).

## Проєктування БД




**Вихідні коди класів системи**

```
public class Theme {
    private final String name;
    private final String backgroundColor;
    private final String textColor;
    private final String keywordColor;
    private final String parameterColor;
    private final String stringColor;
    private final String variableColor;

    public Theme(String name, String backgroundColor, String textColor, String keywordColor,
                 String parameterColor, String stringColor, String variableColor) {
        this.name = name;
        this.backgroundColor = backgroundColor;
        this.textColor = textColor;
        this.keywordColor = keywordColor;
        this.parameterColor = parameterColor;
        this.stringColor = stringColor;
        this.variableColor = variableColor;
    }
```

```
    // --- Гетери для доступу до полів ---
    public String getName() { return name; }
    public String getBackgroundColor() { return backgroundColor; }
    public String getTextColor() { return textColor; }
    public String getKeywordColor() { return keywordColor; }
    public String getParameterColor() { return parameterColor; }
    public String getStringColor() { return stringColor; }
    public String getVariableColor() { return variableColor; }

    // Метод для створення теми за замовчуванням
    public static Theme createDefaultTheme() {
        return new Theme(
            "Default Dark", "#2B2B2B", "#A9B7C6", "#CC7832",
            "#BDB76B", "#6A8759", "#9876AA"
        );
    }
}


public class Settings {
    private Theme activeTheme;
    private String fontName;
    private int fontSize;

    public Settings(Theme activeTheme, String fontName, int fontSize) {
        this.activeTheme = activeTheme;
        this.fontName = fontName;
        this.fontSize = fontSize;
    }
```

```
    // --- Гетери та сеттери ---
    public Theme getActiveTheme() {
        return activeTheme;
    }

    public void setActiveTheme(Theme activeTheme) {
        this.activeTheme = activeTheme;
    }

    public String getFontName() {
        return fontName;
    }

    public void setFontName(String fontName) {
        this.fontName = fontName;
    }

    public int getFontSize() {
        return fontSize;
    }

    public void setFontSize(int fontSize) {
        this.fontSize = fontSize;
    }
}


public class CommandHistoryEntry {
    private final String commandText;
    private final String outputText;
```

```
    private final LocalDateTime timestamp;
    private final boolean isError;

    public CommandHistoryEntry(String commandText, String outputText, boolean isError) {
        this.commandText = commandText;
        this.outputText = outputText;
        this.isError = isError;
        this.timestamp = LocalDateTime.now();
    }

    // --- Гетери ---
    public String getCommandText() { return commandText; }
    public String getOutputText() { return outputText; }
    public LocalDateTime getTimestamp() { return timestamp; }
    public boolean isError() { return isError; }

    @Override
    public String toString() {
        return String.format("[%s] > %s\n%s", timestamp, commandText, outputText);
    }
}




public class TerminalSession {
    private final UUID id;
    private final List<CommandHistoryEntry> history;
    private Process process; // Реальний процес powershell.exe
```

```
public TerminalSession() {
    this.id = UUID.randomUUID();
    this.history = new ArrayList<>();
}

public void start() {
    System.out.println("Запуск сесії " + id);
    // Тут буде логіка запуску процесу powershell.exe через ProcessBuilder
    // try {
    //     ProcessBuilder pb = new ProcessBuilder("powershell.exe", "-NoLogo");
    //     this.process = pb.start();
    // } catch (IOException e) {
    //     e.printStackTrace();
    // }
}

public void executeCommand(String commandText) {
    System.out.println("Сесія " + id + " виконує команду: " + commandText);
    // Тут буде логіка відправки команди в this.process

    // Імітуємо отримання результату
    String output = "Результат виконання '" + commandText + "'";
    boolean isError = false;

    addHistoryEntry(new CommandHistoryEntry(commandText, output, isError));
}

public void addHistoryEntry(CommandHistoryEntry entry) {
    this.history.add(entry);
}
```

```
    public void close() {
        System.out.println("Закриття сесії " + id);
        // Тут буде логіка завершення процесу this.process
        // if (this.process != null) {
        //     this.process.destroy();
        // }
    }

    public UUID getId() {
        return id;
    }

    public List<CommandHistoryEntry> getHistory() {
        return history;
    }
}



public class TerminalApplication {
    private List<TerminalSession> sessions;
    private Settings settings;

    public TerminalApplication() {
        this.sessions = new ArrayList<>();
        loadSettings();
    }

    public void loadSettings() {
```

```
    System.out.println("Завантаження налаштувань...");
    // Тут буде логіка завантаження з БД через Repository
    // Для прикладу, створюємо налаштування за замовчуванням
    Theme defaultTheme = Theme.createDefaultTheme();
    this.settings = new Settings(defaultTheme, "Consolas", 14);
}

public void saveSettings() {
    System.out.println("Збереження налаштувань...");
    // Тут буде логіка збереження в БД через Repository
}

public TerminalSession createSession() {
    TerminalSession newSession = new TerminalSession();
    newSession.start();
    this.sessions.add(newSession);
    System.out.println("Створено нову сесію. Всього сесій: " + sessions.size());
    return newSession;
}

public void closeSession(TerminalSession session) {
    if (session != null) {
        session.close();
        this.sessions.remove(session);
        System.out.println("Закрито сесію. Залишилось сесій: " + sessions.size());
    }
}

public static void main(String[] args) {
    // Простий приклад використання
```

```
        TerminalApplication app = new TerminalApplication();

        TerminalSession session1 = app.createSession();
        session1.executeCommand("Get-Date");
        session1.executeCommand("Get-ChildItem");

        TerminalSession session2 = app.createSession();
        session2.executeCommand("Write-Host 'Hello'");

        app.closeSession(session1);
        app.saveSettings();
    }
}
```

## Питання до лабораторної роботи

1. Що таке UML?

Мова UML – загальноцільова мова візуального моделювання, яка розроблена для специфікації, візуалізації, проєктування та документування компонентів програмного забезпечення, бізнес-процесів та інших систем

2. Що таке діаграма класів UML?

Діаграма класів UML – це структурна діаграма, яка відображає основні сутності системи (класи), їхні атрибути, методи та відносини між ними. Вона використовується для моделювання статичної структури системи і показує, як об’єкти взаємодіють, які дані зберігають і які функції виконують.

Основні елементи:

- Клас – прямокутник із трьома секціями: назва, атрибути, методи.
- Атрибути – характеристики класу (з рівнем доступу: публічний, приватний, захищений).
- Методи (операції) – функції класу.
- Відносини між класами: асоціація, агрегація, композиція, узагальнення (спадкування).

3. Які діаграми UML називають канонічними?
    - Діаграма варіантів використання (Use Case)

- Діаграма класів (Class)
- Діаграма кооперації (Collaboration)
- Діаграма послідовності (Sequence)
- Діаграма станів (Statechart)
- Діаграма діяльності (Activity)
- Діаграма компонентів (Component)
- Діаграма розгортання (Deployment)

4. Що таке діаграма варіантів використання?

Діаграма варіантів використання – 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 варіантів використання складається з кількох основних елементів: прецедентів , акторів та відносин між ними.

5. Що таке варіант використання?

Варіант використання являє собою послідовність дій, який повинен бути виконаний системою, що проєктується при взаємодії її з відповідним актором, самі ці дії не відображаються на діаграмі.

6. Які відношення можуть бути відображені на діаграмі використання?

Існують такі відносини: асоціації, узагальнення, залежність (складається з включення та розширення).

- Асоціація (Association) – узагальнене, невідоме ставлення між актором та варіантом використання.
- Узагальнення (Generalization) – показує, що нащадок успадковує атрибути у свого прямого батьківського елементу
- Залежність (Dependency) – форма взаємозв'язку між двома елементами моделі, призначена для специфікації тієї обставини, що зміна одного елемента моделі призводить до зміни деякого іншого елемента. 25 Загалом залежність є спрямованим бінарним ставленням, яке пов'язує між собою два елементи моделі: незалежний та залежний.
- Відношення включення (include) – окремий випадок загального відношення залежності між двома варіантами використання, при якому деякий варіант використання містить поведінку, визначену в іншому варіанті використання.
- Відношення розширення (extend) – показує, що варіант використання розширює базову послідовність дій та вставляє власну послідовність. У цьому на 26 відміну типу відносин «включення» розширена послідовність може здійснюватися залежно від певних умо

7. Що таке сценарій?

Сценарії використання – це текстові уявлення тих процесів, які відбуваються при взаємодії користувачів системи та самої системи.

8. Що таке діаграма класів?

Діаграма класів (Class Diagram) – Структурна діаграма, що відображає класи системи, їхні атрибути, методи та відносини між ними. Використовується для статичного моделювання системи.

9. Які зв’язки між класами ви знаєте?
    - Асоціація – загальний зв’язок між об’єктами класів.
    - Агрегація – відношення «частина-ціле», де частина може існувати окремо.
    - Композиція – тісніше відношення «частина-ціле», де частина не може існувати без цілого.
    - Узагальнення (спадкування) – зв’язок клас-батько та клас-нащадок.
10. Чим відрізняється композиція від агрегації?
    - Агрегація: частина може існувати окремо від цілого.
    - Композиція: частина існує лише в складі цілого; видалення цілого видаляє частини.
11. Чим відрізняється зв'язки типу агрегації від зв'язків композиції на діаграмах класів?
    - Агрегація – порожній ромб біля цілого.
    - Композиція – заповнений ромб біля цілого.
12. Що являють собою нормальні форми баз даних?

Правила організації даних у таблицях для зменшення надмірності та запобігання логічним помилкам:

- 1НФ – кожен атрибут містить одне значення.
- 2НФ – відсутність часткових залежностей від ключа.
- 3НФ – відсутність транзитивних залежностей.
- BCNF – посилена 3НФ; всі нетривіальні залежності визначені ключами.

13. Що таке фізична модель бази даних? Логічна?

Фізична модель бази даних представляє собор набір бінарних даних у вигляді файлів, структурованих та згрупованих згідно з призначенням (сегменти, екстенти та ін.), що використовується для швидкого та ефективного отримання інформації з жорсткого диска, а також для компактного зберігання та розміщення даних на жорсткому диску.

Логічна модель бази даних є структурою таблиць, уявлень, індексів та інших логічних елементів бази даних, що дозволяють власне програмування та використання бази даних

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

Програмні класи описують сутності системи, а таблиці реалізують їх у БД.

## Висновок

Під час виконання лабораторної роботи я обрати систему побудови UML-діаграм та навчився будувати діаграми варіантів використання для системи що проєктується, розробляти сценарії варіантів використання та будувати діаграми класів предметної області.