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 №2

Технології розроблення програмного забезпечення

«Діаграма варіантів використання. Сценарії варіантів використання. Діаграми UML. Діаграми класів. Концептуальна модель системи»

Виконав
студент групи ІА–34:
Бородай А.С.

Київ 2025

Зміст

Теоретичні відомості

UML є загальноцільова мова візуального моделювання, яка розроблена для специфікації, візуалізації, проектування та документування компонентів програмного забезпечення, бізнес-процесів та інших систем. Мова UML ефективно використовується для побудови концептуальних, логічних та графічних моделей складних систем різного цільового призначення.

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. Основними елементами є: варіанти використання або прецеденти (use case), актор або дійова особа (actor) та відносини між акторами та варіантами використання (relationship).

Актор (**actor**) – будь-який об'єкт, суб'єкт чи система, що взаємодіє з модельованою бізнес-системою ззовні для досягнення своїх цілей або вирішення

певних завдань. Це може бути людина, технічний пристрій, програма або будь-яка інша система, яка служить джерелом впливу на систему, що моделюється.

Варіант використання (**use case**) служить для опису служб, які система надає актору. Інакше кажучи кожен варіант використання визначає набір дій, здійснюваний системою під час діалогу з актором. Кожен 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.

Відношення (**relationship**) – семантичний зв'язок між окремими елементами

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

Асоціація (**association**) – узагальнене, невідоме ставлення між актором та варіантом використання. Позначається суцільною лінією між актором та варіантом використання. Розрізняють ненаправлену (двонаправлену) асоціацію та однонаправлену асоціацію. Ненаправлена асоціація показує взаємодію без акцента на напрямок, або коли напрямок ще не аналізувався.

Спрямована, або направлена асоціація (**directed association**) – також показує що актор асоціюється з варіантом використання але показує, що варіант використання ініціалізується актором. Спрямована асоціація дозволяє запровадити поняття основного актора (він є ініціатором асоціації) та другорядного актора (варіант використання є ініціатором, тобто передає акторові довідкові відомості або звіт про виконану роботу).

Відношення узагальнення (**generalization**) – показує, що нащадок успадковує атрибути у свого прямого батьківського елементу. Тобто, один елемент моделі є спеціальним або окремим випадком іншого елемента моделі. Може застосовуватися як до акторів, так і до варіантів використання.

Відношення залежності (**dependency**) визначається як форма взаємозв'язку між двома елементами моделі, призначена для специфікації тієї обставини, що зміна одного елемента моделі призводить до зміни деякого іншого елемента.

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 відміну типу відносин «включення» розширена послідовність може здійснюватися залежно від певних умов. Графічне зображення відношення розширення – пунктирна стрілка спрямована від залежного варіанта (розширює) до незалежного варіанта (базового) з ключовим словом <<extend>>.

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.

Клас – це основний будівельний блок програмної системи. 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.

Атрибути класу визначають склад та структуру даних, що зберігаються в об'єктах цього класу. Кожен атрибут має ім'я та тип, який визначає, які дані він представляє.

• + Відкритий (**public**) – атрибут видно для будь-якого іншого класу (об'єкта);
• ~ В межах пакету (**package**) – атрибут видно в цьому ж пакеті;
• # Захищений (**protected**) – атрибут видно для нащадків цього класу;
• - Закритий (**private**) – може використовуватися лише об'єктом, що його містить.

Відносини між класами: асоціації та узагальнення.

Асоціація – найбільш загальний вид зв'язку між двома класами системи. Як правило, вона відображає використання одного класу іншим за допомогою певної якості або поля.

Узагальнення (успадкування) на діаграмах класів використовується, щоб показати зв'язок між класом-батьком та класом-нащадком.

Агрегацією позначається відношення частина-ціле, коли об'єкти одного класу входять до об'єкта іншого класу. У цьому випадку список буде виступати агрегатом, а об'єкти, що входять до списку, елементами, що агрегуються.

Композицією також позначається відношення частина-ціле, але позначає тісніший зв'язок між елементами, що представляють ціле та частини.

Якщо говорити про відображення агрегації та композиції в програмному коді, то агрегація, як правило відображується як посилання на об’єкт, а композиція – це змінна типу структури. Біля цілого може бути вказана назва, яка вказує на назву поля класу, яке відображений на діаграмі цим відношенням.

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.

Хід роботи

Обрана тема: Mind-mapping software

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).

Зробивши аналіз предметної області та функціональних вимог до застосунку, було визначено акторів та сценарії використання.

UseCase Diagram

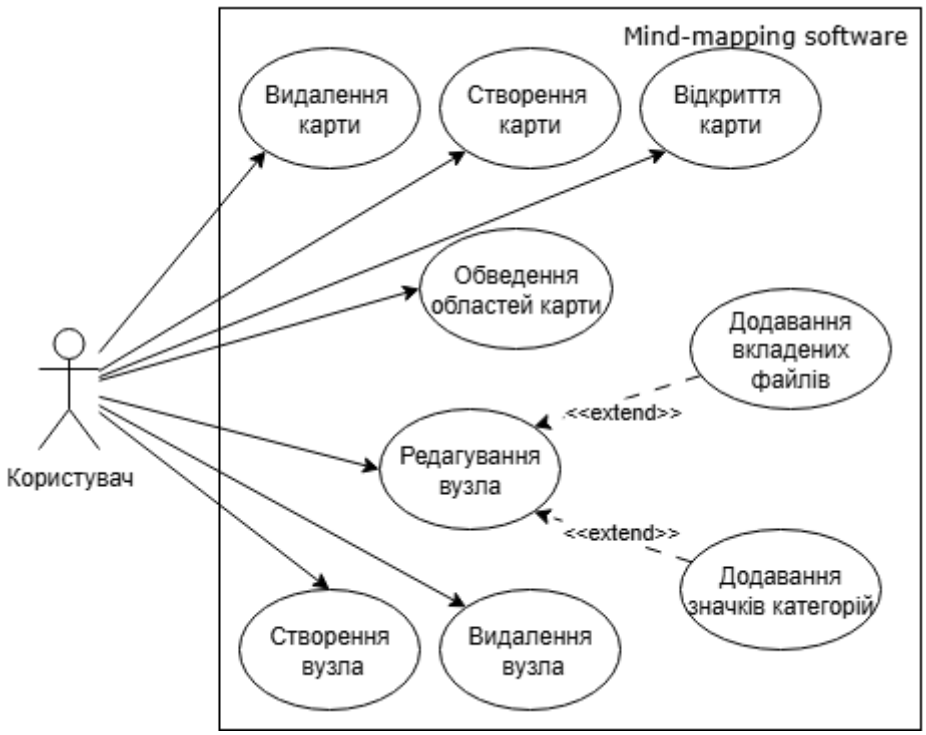

Рисунок 1 – UseCase Diagram.

Сценарії використання

Варіант використання “Додавання значків категорій”:

Передумови: користувач створив вузол.

Постумови: вузол змінив параметр терміновості.

Взаємодіючі сторони: користувач, MindMapping Software.

Короткий опис: користувач додає параметр вузла “терміновість”.

Основний перебіг подій:

1. Користувач натискає на кнопку “Редагувати”
2. Система відкриває діалог редагування вузла.
3. Користувач додає значок.
4. Система редагує вузол.
5. Відображається відредагований вузол.

Винятки: немає.

Примітки: немає.

Варіант використання “Створення вузла”:

Передумови: користувач відкрив карту або створив нову.

Постумови:

- Новий вузол створено й відображено на полотні.
- Вузол збережено у репозиторії.

Взаємодіючі сторони: користувач, MindMapping Software.

Короткий опис: користувач додає новий вузол до карти, щоб зафіксувати нову ідею.

Основний перебіг подій:

1. Користувач натискає правою кнопкою миші на полотні.
2. З’являється контекстне меню з опцією "Додати вузол".
3. Користувач обирає цю опцію.
4. Система створює новий вузол у вказаній позиції.
5. Вузол додається до карти і відображається на полотні.

Винятки: карта не відкрита.

Примітки: вузол може мати текст за замовчуванням ("Нова ідея").

Варіант використання “Видалення вузла”:

Передумови:

- Відкрита карта.
- У карті існує хоча б один вузол.

Постумови:

- Вибраний вузол і всі його з’єднання видалені.
- Зміни збережені у репозиторії.

Взаємодіючі сторони: користувач, MindMapping Software.

Короткий опис: користувач видаляє непотрібний вузол разом з його зв’язками.

Основний перебіг подій:

1. Користувач виділяє вузол на полотні.
2. Викликає команду "Видалити".
3. Система перевіряє, чи вузол має з’єднання.
4. Видаляє вузол і всі відповідні з’єднання.
5. Оновлює відображення карти.

Винятки: немає.

Примітки: видалення вузла є незворотнім

Діаграми класів використовуються при моделюванні програмних систем найчастіше. Вони є однією із форм статичного опису системи з погляду її проєктування, показуючи її структуру.

Class Diagram

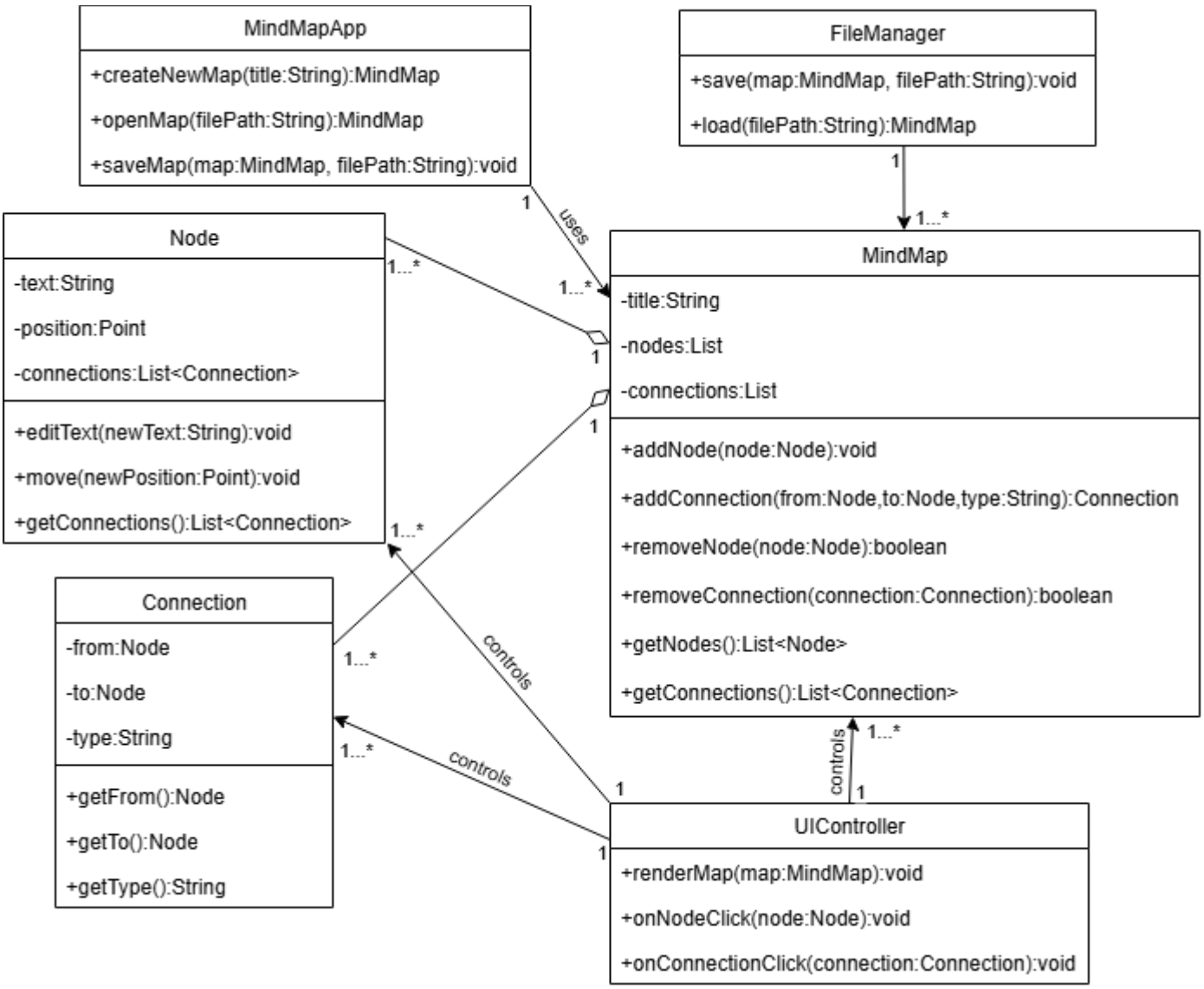

Рисунок 2 – Class Diagram.

Вихідні коди класів

Клас MindMapApp

```
class MindMapApp {
    public MindMap createNewMap(String title);
    public MindMap openMap(String filePath);
    public void saveMap(MindMap map, String filePath);
}
```

Клас MindMap

```
@Entity
@Table(name = "mindmaps")
class MindMap {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;
        private String title;
        @OneToMany(mappedBy = "mindMap", cascade = CascadeType.ALL)
        private List<Node> nodes;
        @OneToMany(mappedBy = "mindMap", cascade = CascadeType.ALL)
        private List<Connection> connections;

        public void addNode(Node node);
        public boolean removeNode(Node node);
        public Connection addConnection(Node from, Node to, String type);
        public boolean removeConnection(Connection connection);
        public List<Node> getNodes();
        public List<Connection> getConnections();
}
```

Клас Node

```
@Entity
@Table(name = "nodes")
class Node {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;
	private String text;
	@ManyToOne
	@JoinColumn(name = "mindmap_id")
	private MindMap mindMap;
	private Point position;
	@OneToMany(mappedBy = "target", cascade = CascadeType.ALL)
	private List<Connection> connections;

	public void editText(String newText);
	public void move(Point newPosition);
	public List<Connection> getConnections();
}
```

Клас Connection

```
@Entity
@Table(name = "connections")
class Connection {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;
	@ManyToOne
	@JoinColumn(name = "mindmap_id")
	private MindMap mindMap;
	@ManyToOne
```

```
    @JoinColumn(name = "source_id")
    private Node from;
    @ManyToOne
    @JoinColumn(name = "target_id")
    private Node to;
    private String type;

    public Node getFrom();
    public Node getTo();
    public String getType();
}
```

Клас FileManager

```
class FileManager {
    public void save(MindMap map, String filePath);
    public MindMap load(String filePath);
}
```

Клас UIController

```
class UIController {
    public void renderMap(MindMap map);
    public void onNodeClick(Node node);
    public void onConnectionClick(Connection connection);
}
```

Структура бази даних

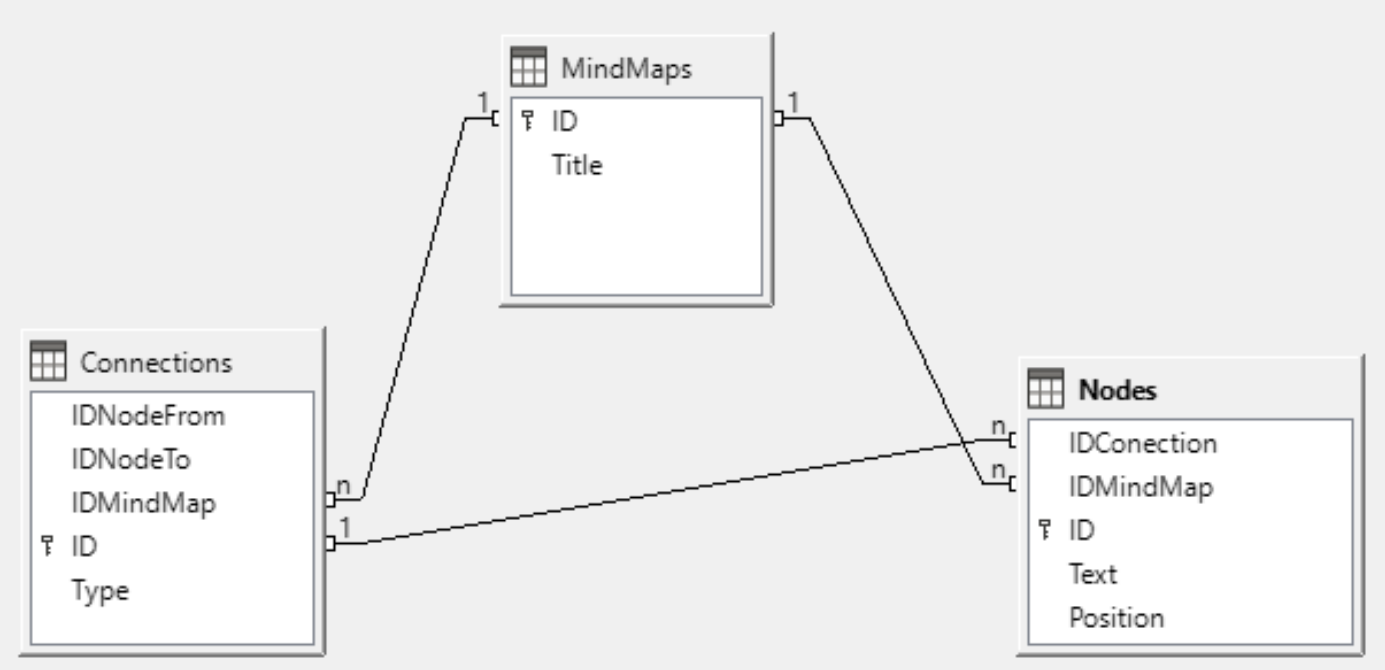

Рисунок 3 – Структура бази даних.

Висновок

У ході виконання лабораторної роботи було проаналізовано предметну область “MindMap”. Було визначено основний функціонал та можливості системи. На основі функціональних можливостей було створено UML діаграму застосувань та класів. Діаграма класів використалась для створення основної структури бази даних, що включає необхідний мінімум таблиць для роботи.

Також було описано декілька варіантів застосувань нашого застосунку, що допомогло доповнити наше уявлення про використання застосунку нашим користувачем.