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

«Основи проектування»

Виконав
студент групи ІА-34:
Сльота Максим

Перевірив:
Мягкий М. Ю.

Київ - 2025

Тема: Основи проектування

Мета роботи: 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

Теоретичні відомості:

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

Рівень представлення (layer) – спосіб організації та розгляду моделі на одному рівні абстракції, що представляє горизонтальний зріз архітектури моделі, тоді як розбиття представляє її вертикальний зріз.

Діаграма (diagram) – графічне уявлення сукупності елементів моделі у формі зв'язкового графа, вершинам і ребрам (дугам) якого приписується певна семантика.

Діаграма варіантів використання (Use-Cases Diagram) – це UML діаграма за допомогою якої у графічному вигляді можна зобразити вимоги до системи, що розробляється.

Відношення (relationship) – семантичний зв'язок між окремими елементами моделі. Один актор може взаємодіяти з кількома варіантами використання.

Асоціація (association) – узагальнене, невідоме ставлення між актором та варіантом використання.

Відношення узагальнення (generalization) – показує, що нащадок успадковує атрибути у свого прямого батьківського елементу.

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.

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

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

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.

Для кожного атрибуту класу можна встановити видимість (visibility). Ця характеристика показує, чи доступний атрибут для інших класів. У UML визначено такі рівні видимості атрибутів:

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

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.

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

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

Нормальна форма – властивість відношення в реляційній моделі даних, що характеризує його з погляду надмірності, що потенційно призводить до логічно помилкових результатів вибірки або зміни даних.

Обрана тема: System activity monitor

Монітор активності системи повинен зберігати і запам'ятовувати статистику використовуваних компонентів системи, включаючи навантаження на процесор, обсяг займаної оперативної пам'яті, натискання клавіш на клавіатурі, дії миші (переміщення, натискання), відкриття вікон і зміна вікон; будувати звіти про використання комп'ютера за різними критеріями (% часу перебування у веб-браузері, середнє навантаження на процесор по годинах, середній час роботи комп'ютера по днях і т.д.); правильно поводитися з «простоюванням» системи – відсутністю користувача.

Хід роботи:

1. Діаграма варіантів використання

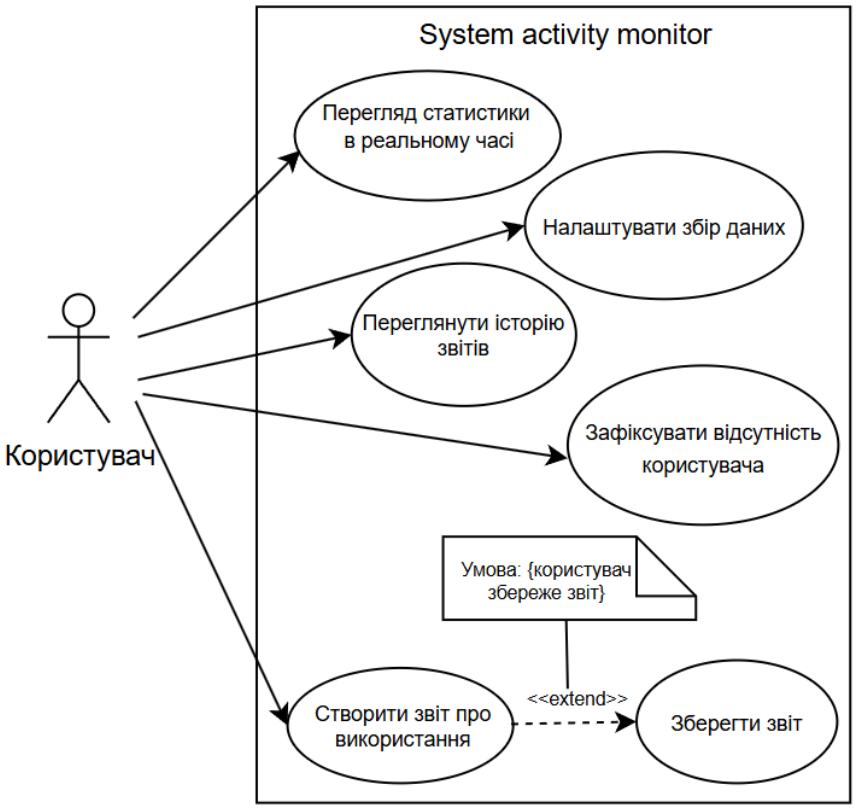

Рис 1. Діаграма варіантів використання

2. Діаграма класів предметної області

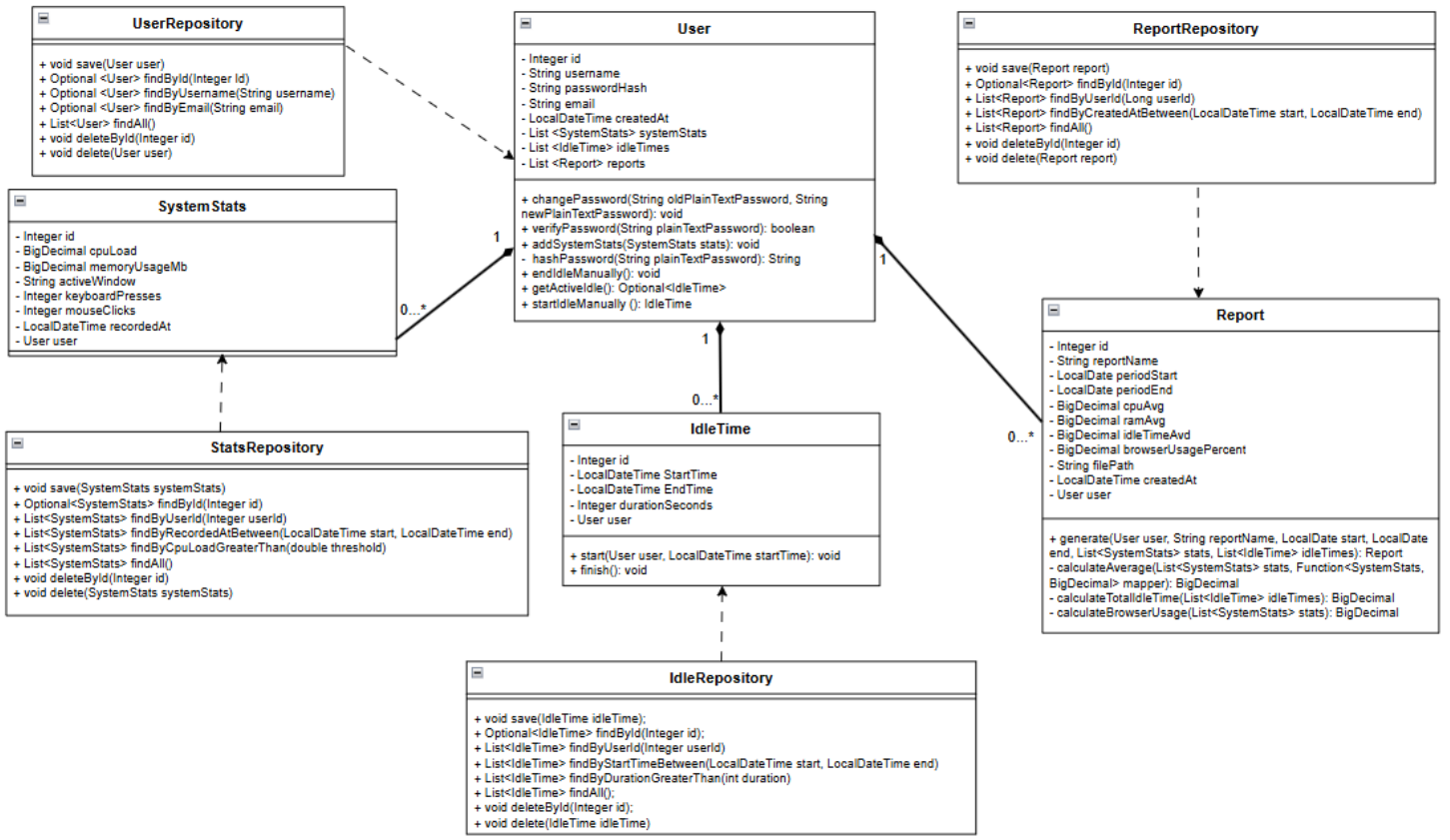

Рис 2. Діаграма класів

3. Три сценарії варіантів використання

Варіант використання 1: Перегляд статистики в реальному часі

Передумови: Користувач успішно авторизувався в системі.

Постумови: Користувач отримує доступ до актуальної інформації про роботу системи (CPU, RAM, введення з клавіатури та миші, активність вікон, простої системи).

Взаємодіючі сторони: Звичайний користувач, Система моніторингу.

Короткий опис: Користувач може переглянути поточні показники активності системи в режимі реального часу.

Основний потік подій:

1. Користувач обирає функцію «Перегляд статистики в реальному часі».
2. Система збирає поточні дані з моніторингу ресурсів, введення користувача, активності вікон і стану простою.
3. Система відображає статистику у вигляді числових показників, діаграм або графіків.
4. Користувач переглядає актуальні дані.

Винятки:

Виняток №1: Система не може отримати деякі дані. Виводиться повідомлення про помилку, пропонується повторити запит.

Примітки: Дані оновлюються автоматично з певною частотою (наприклад, кожні 5 секунд).

Варіант використання 2: Створити звіт про використання

Передумови: Користувач авторизувався в системі. Є накопичена статистика за обраний період.

Постумови: Система формує звіт, який можна зберегти або переглянути пізніше.

Взаємодіючі сторони: Звичайний користувач, Система моніторингу.

Короткий опис: Користувач може створити звіт про використання системи за обраними параметрами.

Основний потік подій:

1. Користувач обирає функцію «Створити звіт про використання».
2. Система запитує параметри звіту (період, типи даних, формат).
3. Користувач задає параметри.
4. Система формує звіт на основі накопичених статистичних даних.
5. Система повідомляє про успішне створення звіту.

Винятки:

Виняток №1: Недостатньо даних за вибраний період — система повідомляє користувача та пропонує змінити параметри.

Виняток №2: Помилка при формуванні звіту — система пропонує повторити операцію.

Примітки: Створений звіт може бути збережений користувачем — опція «Зберегти звіт».

Варіант використання 3: Зберегти звіт

Передумови: Користувач створив звіт про використання системи.

Постумови: Звіт збережено у базі даних або експортовано у вибраному форматі.

Взаємодіючі сторони: Звичайний користувач, Система моніторингу.

Короткий опис: Користувач може зберегти створений звіт локально або у системі.

Основний потік подій:

1. Користувач після створення звіту обирає функцію «Зберегти звіт».
2. Система пропонує вибрати місце збереження або формат файлу.
3. Користувач підтверджує дію.
4. Система зберігає звіт і повідомляє про успішне завершення.

Винятки:

Виняток №1: Помилка запису файлу — система виводить повідомлення та пропонує повторити спробу.

Примітки: Цей варіант використання є розширенням сценарію «Створити звіт про використання».

4. Вихідний код класів системи

User.java:

```
@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(nullable = false, unique = true, length = 50)
    private String username;

    @Column(name = "password_hash", nullable = false, length = 255)
    private String passwordHash;

    @Column(length = 100)
    private String email;

    @Column(name = "created_at", nullable = false, updatable = false)
    private final LocalDateTime createdAt;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true, fetch =
FetchType.LAZY)
    private final List<SystemStats> systemStats = new ArrayList<>();

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true, fetch =
FetchType.LAZY)
    private final List<IdleTime> idleTimes = new ArrayList<>();

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true, fetch =
FetchType.LAZY)
    private final List<Report> reports = new ArrayList<>();

public void changePassword(String oldPlainTextPassword, String newPlainTextPassword) {}
public boolean verifyPassword(String plainTextPassword) {}
public void addSystemStats(SystemStats stats) {}
private String hashPassword(String plainTextPassword) {}
```

SystemStats.java:

```
@Entity
@Table(name = "system_stats")
public class SystemStats {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @Column(name = "cpu_load", precision = 5, scale = 2, nullable = false)
    private final BigDecimal cpuLoad;

    @Column(name = "memory_usage_mb", precision = 10, scale = 2, nullable = false)
    private final BigDecimal memoryUsageMb;

    @Column(name = "active_window", length = 255)
    private final String activeWindow;

    @Column(name = "keyboard_presses")
    private final Integer keyboardPresses;

    @Column(name = "mouse_clicks")
    private final Integer mouseClicks;

    @Column(name = "recorded_at", updatable = false)
    private final LocalDateTime recordedAt;
```

IdleTime.java:

```
@Entity
@Table(name = "idle_time")
public class IdleTime {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @Column(name = "start_time", nullable = false, updatable = false)
    private LocalDateTime startTime;
```

```
    @Column(name = "end_time")
    private LocalDateTime endTime;

    @Column(name = "duration_seconds")
    private Integer durationSeconds;

public void start(User user) {}
public void finish() {}
```

Report.java:

```
@Entity
@Table(name = "reports")
public class Report {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @Column(name = "report_name", nullable = false, length = 255)
    private String reportName;

    @Column(name = "period_start", nullable = false)
    private LocalDate periodStart;

    @Column(name = "period_end", nullable = false)
    private LocalDate periodEnd;

    @Column(name = "cpu_avg", precision = 5, scale = 2)
    private BigDecimal cpuAvg;

    @Column(name = "ram_avg", precision = 10, scale = 2)
    private BigDecimal ramAvg;

    @Column(name = "idle_time_total_seconds", precision = 10, scale = 2)
    private BigDecimal idleTimeTotalSeconds;

    @Column(name = "browser_usage_percent", precision = 5, scale = 2)
    private BigDecimal browserUsagePercent;

    @Column(name = "file_path", length = 500)
    private String filePath;

    @Column(name = "created_at", updatable = false)
    private LocalDateTime createdAt;
```

```
public static Report generate(User user, String reportName, LocalDate start, LocalDate end,
List<SystemStats> stats, List<IdleTime> idleTimes) {}
private static BigDecimal calculateAverage(List<SystemStats> stats, Function<SystemStats, BigDecimal>
mapper) {}
private static BigDecimal calculateTotalIdleTime(List<IdleTime> idleTimes) {}
private static BigDecimal calculateBrowserUsage(List<SystemStats> stats) {}
```

5. Структура база даних

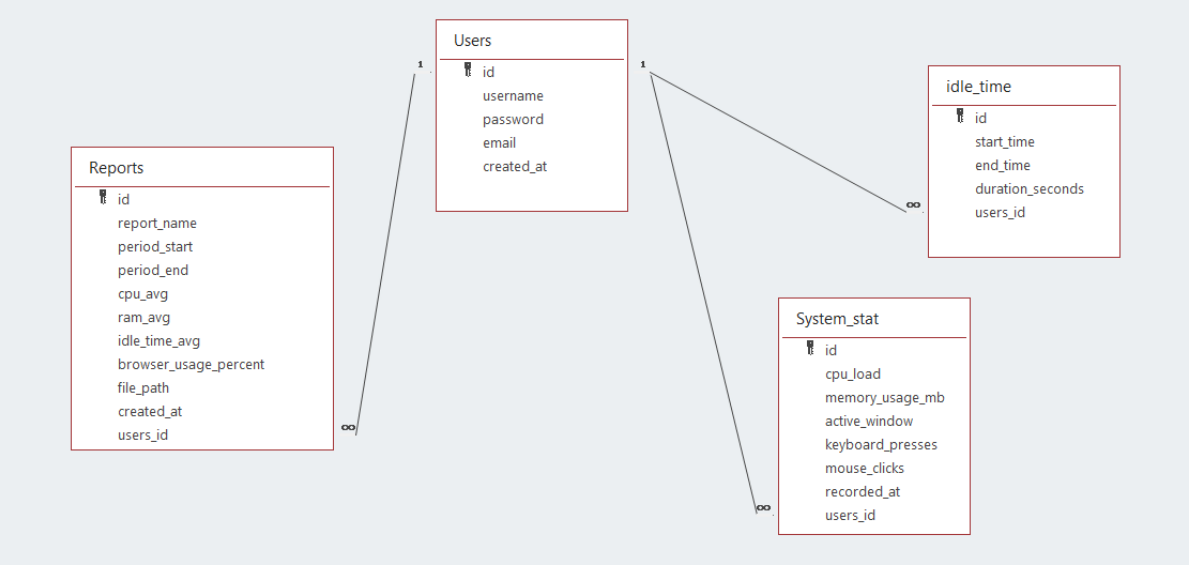

Рис 3. Структура база даних

Висновок: Під час виконання лабораторної роботи було проаналізовано предметну область монітора активності системи, на основі чого створено діаграму варіантів використання для моделювання взаємодії користувача з програмою. Спроєктовано діаграму класів, яка стала основою для програмної реалізації. Відповідно до цієї діаграми розроблено основні класи та структуру бази даних. Для ефективної взаємодії між моделлю та базою даних було застосовано шаблон Repository.

Відповідь на контрольні питання:

1. Що таке UML?

UML (Unified Modeling Language) — це уніфікована мова моделювання, яка використовується в об'єктно-орієнтованому програмуванні для візуалізації, специфікації, конструювання та документування програмних систем. UML надає стандартний спосіб представлення архітектури системи, включаючи такі елементи, як дії, компоненти, класи та їхні взаємозв'язки. Це, по суті, набір графічних нотацій, що допомагає розробникам, аналітикам та іншим зацікавленим сторонам краще розуміти та обговорювати проєкт.

2. Що таке діаграма класів UML?

Діаграма класів — це статична структурна діаграма UML, яка описує структуру

системи, показуючи класи системи, їхні атрибути, операції (або методи) та відношення між класами. Вона є однією з найважливіших і найчастіше використовуваних діаграм в UML, оскільки дає чітке уявлення про будову програмної системи.

3. Які діаграми UML називають канонічними?

Ці діаграми були виділені "Групою трьох" (Граді Буч, Івар Якобсон, Джеймс Рамбо) як ключові для об'єктно-орієнтованого аналізу та проєктування:

1. Діаграма варіантів використання (Use-Case Diagram): Описує функціональність системи з точки зору користувача.
2. Діаграма класів (Class Diagram): Показує статичну структуру системи.
3. Діаграма послідовності (Sequence Diagram): Демонструє взаємодію об'єктів у часі.
4. Діаграма станів (Statechart Diagram): Моделює динамічну поведінку об'єкта протягом його життєвого циклу.
5. Діаграма діяльності (Activity Diagram): Ілюструє потік робіт або операцій.

4. Що таке діаграма варіантів використання?

Діаграма варіантів використання (Use-Case Diagram) — це діаграма UML, що описує взаємодію між користувачами (акторами) та системою. Вона показує, які функції (варіанти використання) система надає акторам. Головна мета цієї діаграми — визначити та візуалізувати функціональні вимоги до системи.

5. Що таке варіант використання?

Варіант використання (Use Case) — це опис послідовності дій, які виконує система у відповідь на запит від актора для досягнення певної мети. По суті, це окрема функція системи. Наприклад, у системі інтернет-магазину варіантами використання можуть бути "Пошук товару", "Додавання товару до кошика", "Оформлення замовлення".

6. Які відношення можуть бути відображені на діаграмі використання?

На діаграмі варіантів використання існують такі основні типи відношень:

- Асоціація (Association): Зв'язок між актором та варіантом використання. Показує, що актор взаємодіє з даною функцією системи.
- Включення (Include): Показує, що один варіант використання обов'язково включає в себе функціональність іншого. Це допомагає уникнути дублювання коду.
- Розширення (Extend): Визначає, що один варіант використання може за певних умов доповнювати функціональність іншого. Це опціональна поведінка.
- Узагальнення (Generalization): Вказує, що один актор або варіант використання є узагальненням (батьком) для іншого (нащадка). Нащадок успадковує

характеристики батька і може додавати власні.

7. Що таке сценарій?

Сценарій — це конкретний екземпляр варіанту використання, який описує одну з можливих послідовностей подій. Кожен варіант використання може мати кілька сценаріїв. Зазвичай виділяють:

- Основний (успішний) сценарій: Описує ідеальний хід подій, коли все йде за планом і мета досягається.
- Альтернативні сценарії: Описують інші, менш поширені шляхи виконання або обробку помилок та виняткових ситуацій.

8. Що таке діаграма класів?

Діаграма класів — це статична діаграма, яка описує структуру системи. Вона моделює класи, їх атрибути (дані) та методи (поведінку), а також різноманітні статичні зв'язки між цими класами.

9. Які зв’язки між класами ви знаєте?

Існують наступні основні типи зв'язків між класами в UML:

- Асоціація (Association): Найзагальніший тип зв'язку, що показує, що об'єкти одного класу якимось чином пов'язані з об'єктами іншого.
- Агрегація (Aggregation): Спеціальний випадок асоціації типу "частина-ціле", де об'єкти-частини можуть існувати незалежно від об'єкта-цілого.
- Композиція (Composition): Більш сувора форма агрегації, де об'єкти-частини не можуть існувати без об'єкта-цілого. Якщо "ціле" знищується, то і його "частини" теж.
- Успадкування або Узагальнення (Inheritance/Generalization): Відношення "є різновидом" (is-a), де один клас (підклас) успадковує властивості та методи іншого класу (суперкласу).
- Залежність (Dependency): Показує, що зміна в одному класі (незалежному) може вплинути на інший клас (залежний), але зв'язок між ними не є постійним.

10. Чим відрізняється композиція від агрегації?

Основна відмінність полягає у життєвому циклі об'єктів:

- Агрегація: Це слабкий зв'язок "частина-ціле". "Частина" може існувати незалежно від "цілого". Наприклад, автомобіль та його пасажири. Пасажири можуть існувати і без автомобіля.
- Композиція: Це сильний зв'язок "частина-ціле". "Частина" не може існувати без

"цілого". Життєвий цикл "частини" повністю залежить від "цілого". Наприклад, будинок та його кімнати. Кімнати не можуть існувати окремо від будинку; якщо будинок зруйновано, кімнат теж не стане.

11. Чим відрізняється зв'язки типу агрегації від зв'язків композиції на діаграмах класів?

На діаграмах класів UML агрегація та композиція позначаються лінією асоціації, але з різними ромбами на кінці з боку "цілого":

- Агрегація: Позначається незафарбованим (пустим) ромбом.
- Композиція: Позначається зафарбованим (чорним) ромбом.

12. Що являють собою нормальні форми баз даних?

Нормальні форми (НФ) — це набір правил і вимог до структури реляційної бази даних, метою яких є зменшення надлишковості даних та усунення аномалій (помилок) при їх оновленні, додаванні чи видаленні. Процес приведення бази даних до цих вимог називається нормалізацією.

Основні нормальні форми:

- Перша нормальна форма (1НФ): Усі атрибути (комірки таблиці) повинні бути атомарними, тобто містити неподільні значення. Кожен запис має бути унікальним.
- Друга нормальна форма (2НФ): Таблиця повинна бути в 1НФ, і всі неключові атрибути повинні повністю залежати від первинного ключа (якщо ключ складений).
- Третя нормальна форма (3НФ): Таблиця повинна бути в 2НФ, і не повинно бути транзитивних залежностей (коли неключовий атрибут залежить від іншого неключового атрибута).
- Нормальна форма Бойса-Кодда (НФБК): Більш сувора версія 3НФ.

13. Що таке фізична модель бази даних? Логічна?

Це два рівні представлення структури бази даних:

- Логічна модель даних: Описує дані та зв'язки між ними незалежно від конкретної системи управління базами даних (СУБД). Вона фокусується на бізнес-вимогах і структурі даних з точки зору користувача (наприклад, сутності, атрибути, зв'язки). Прикладом є ER-діаграма (Entity-Relationship).
- Фізична модель даних: Описує, як дані будуть фізично зберігатися в конкретній СУБД. Вона включає такі деталі, як назви таблиць та стовпців, типи даних для

кожного стовпця, індекси, обмеження цілісності та інші специфічні для СУБД параметри.

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

Існує дуже тісний зв'язок, особливо в об'єктно-орієнтованому програмуванні. Цей зв'язок реалізується за допомогою технології ORM (Object-Relational Mapping).

Основні принципи відповідності:

- Клас і Таблиця: Кожен клас у програмі зазвичай відповідає таблиці в базі даних.
- Об'єкт і Рядок: Екземпляр класу (об'єкт) відповідає одному рядку (запису) в таблиці.
- Атрибут класу і Стовпець таблиці: Поля або властивості класу відповідають стовпцям у таблиці.
- Відношення між класами і Зв'язки між таблицями:
    - Асоціація "один до багатьох" реалізується через зовнішній ключ (foreign key).
    - Асоціація "багато до багатьох" реалізується через проміжну (зв'язуючу) таблицю.