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*

із дисципліни ***«Технології розробки програмного забезпечення»***

Тема: ***«Основи проектування»***

Виконав:

Студент групи ІА-34

Ястремський Богдан

Перевірив:

асистент кафедри ІСТ

Мягкий Михайло Юрійович

Київ — 2025

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.

Короткі 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.

Є 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. Вона не описує внутрішню побудову системи.

Актор - людина, технічний пристрій, програма або будь-яка інша система, яка служить джерелом впливу на систему, що моделюється.

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

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

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.

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

Хід роботи:

1. Проаналізувати тему та спроєктувати діаграму варіантів використання відповідно до обраної теми лабораторного циклу.

   1.1. Аналіз теми.

   Web-browser (proxy, chain of responsibility, factory method, template method, visitor, p2p). Веб-браузер повинен мати можливість зробити наступне: мати адресний рядок для введення адреси сайту, переміщатися і відображати структур html документа, переглядати підключений javascript та css файли, перегляд всіх підключених ресурсів (зображень), коректна обробка відповідей з сервера (коди відповідей HTTP) – переходи при перенаправленнях, відображення сторінок 404 і 502/503.

   1.2. Проектування діаграми варіантів використання.

   Актори: **Користувач** (головний, взаємодіє з браузером), **Веб-браузер** (головна система), **Веб-сервер** (відповідає на запити браузера).

   Варіанти використання:

   - Введення URL - Користувач вводить URL у адресний рядок.
   - Завантаження сторінки - Браузер робить запит на сервер і отримує HTML документ.
   - Переміщення по сторінці - Користувач переміщається по вже завантаженій сторінці.
   - Перегляд HTML документа - Користувач переглядає вихідний код HTML документу.

- Перегляд CSS та JavaScript - Користувач переглядає підключені CSS і JavaScript файли.
- Перегляд зображень та інших ресурсів - Користувач переглядає зображення та інші ресурси, що завантажуються.
- Обробка HTTP відповіді - Браузер обробляє HTTP відповіді (наприклад, 200, 404, 502).
- Перехід при перенаправленні - Браузер обробляє статуси 301/302 і здійснює автоматичний перехід.
- Відображення сторінки з помилкою - Браузер показує сторінку помилки (наприклад, 404 чи 502).

Зв’язки між акторами і варіантами використання:

1. Користувач - Введення URL (напрямлена асоціація) Користувач ініціює дію введення URL в адресний рядок.
2. Користувач - Переміщення по сторінці (напрямлена асоціація). Користувач може переміщатися по сторінці після її завантаження.
3. Користувач - Перегляд HTML документа (напрямлена асоціація). Користувач може захотіти побачити HTML код завантаженої сторінки.
4. Користувач - Перегляд CSS та JavaScript (напрямлена асоціація). Користувач може перевірити або проаналізувати стилі та скрипти на сторінці.
5. Користувач - Перегляд зображень та інших ресурсів (напрямлена асоціація). Користувач може переглядати ресурси на сторінці, такі як зображення, відео тощо.
6. Веб-сервер - Завантаження сторінки (напрямлена асоціація) Веб-сервер надає браузеру HTML документ у відповідь на запит.

7. Веб-сервер - Перехід при перенаправленні (напрямлена асоціація). Веб-сервер відправляє код перенаправлення (301, 302), і браузер виконує новий запит.
8. Веб-сервер - Обробка HTTP відповіді (напрямлена асоціація) Веб-сервер відповідає браузеру HTTP статусами (наприклад, 200, 404, 502), а браузер обробляє ці відповіді.
9. Веб-сервер - Відображення сторінки з помилкою (напрямлена асоціація). Веб-сервер може відправити код помилки (404, 502), і браузер відображає відповідну сторінку помилки.
10. Браузер - Обробка HTTP відповіді (напрямлена асоціація) Браузер обробляє відповіді, отримані від сервера, і здійснює відповідні дії.
11. Обробка HTTP відповіді - Перехід при перенаправленні (include). Обробка перенаправлення є частиною загальної обробки HTTP відповіді.
12. Обробка HTTP відповіді - Відображення сторінки з помилкою (extend). Якщо отримано код помилки, обробка цього коду розширює стандартну поведінку обробки відповіді і показує сторінку помилки.

1.3. Власне діаграма:

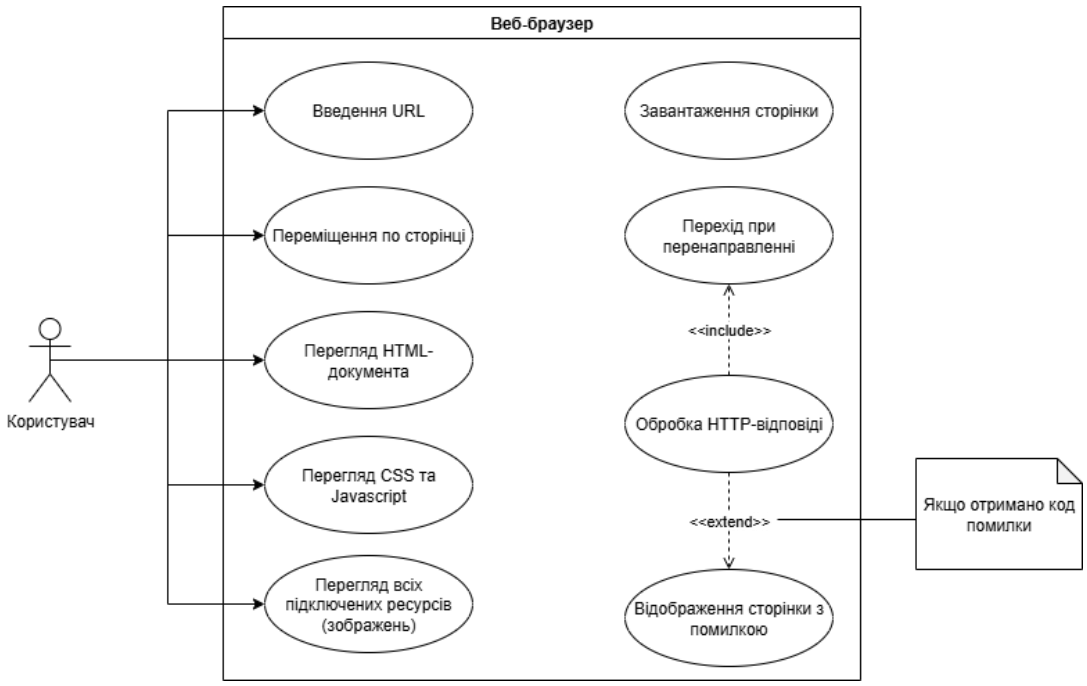

Рис. 1 – Діаграма варіантів використання системи

2. Спроєктувати діаграму класів предметної області.

2.1. Browser (Браузер)

Опис: Основний клас, що керує взаємодією з користувачем, організовує завантаження сторінок, взаємодію з іншими класами.

Атрибути:

- addressBar: AddressBar (адресний рядок).
- currentPage: WebPage (поточна веб-сторінка).

Методи:

- loadPage(url: String): завантажує веб-сторінку за вказаним URL.
- displayPage(page: WebPage): відображає веб-сторінку.
- handleError(errorCode: Int): обробляє помилки (404, 502 і т.д.).

2.2. AddressBar (Адресний рядок)

Опис: Клас для управління введенням URL і запитом на завантаження веб-сторінки.

Атрибути:

- url: String рядок, що містить введену адресу.
- protocol: String протокол адреси.

Методи:

- getInput(): String отримує введену URL-адресу.
- validateURL(url: String): Boolean перевіряє правильність введеного URL.

2.3. WebPage (Веб-сторінка)

Опис: Клас, що містить інформацію про HTML-сторінку, підключені ресурси та їхній вміст.

Атрибути:

- htmlResources: List<HTMLFile> (HTML файли).
- cssResources: List<CSSFile> (CSS файли, підключені до сторінки).
- jsResources: List<JSFile> (JavaScript файли).
- imageResources: List<ImageFile> (зображення).

Методи:

- parseHTML(): парсить HTML-код і витягує відповідні елементи.
- loadResources(): завантажує всі підключені ресурси (CSS, JavaScript, зображення).

2.4. HTMLFile

Опис: Клас, що містить інформацію про html-файл:

Атрибути:

- fileName: String назва файлу
- filePath: String шлях до файлу
- content: String вміст файлу

Методи:

- loadHTML: завантажити вміст файлу

2.5. CSSFile

Опис: Клас, що містить інформацію про css-файл:

Атрибути:

- fileName: String назва файлу
- filePath: String шлях до файлу
- content: String вміст файлу

Методи:

- loadCSS: завантажити вміст файлу

2.6. JSFile

Опис: Клас, що містить інформацію про js-файл:

Атрибути:

- fileName: String назва файлу
- filePath: String шлях до файлу
- content: String вміст файлу

Методи:

- loadJS: завантажити вміст файлу

2.7. ImageFile

Опис: Клас, що містить інформацію про файл-картинку:

Атрибути:

- fileName: String назва файлу
- filePath: String шлях до файлу
- content: String вміст файлу

Методи:

- loadImage: завантажити вміст файлу

2.8. WebServer

Опис: Клас, що містить інформацію про файл-картинку:

Атрибути:

- host: String зберігає ім’я хоста або адресу сервера.
- resources: List<String> список ресурсів, доступних на сервері
- pages: List<WebPage> список веб-сторінок, які сервер може відправляти у відповідь на запит.

Методи:

- processRequest(request: HTTPRequest): HTTPResponse обробляє HTTP запит і генерує HTTP відповідь
- sendResponse(response: HTTPResponse) відправляє відповідь клієнту

2.9. HTTPRequest:

Опис: Клас, що відповідає за відправлення запиту до сервера і отримання відповіді.

Атрибути:

- url: String адреса ресурсу.
- method: String — метод HTTP (наприклад, GET, POST).

Методи:

- sendRequest(): HTTPResponse відправляє запит на сервер і отримує HTTPResponse.
- parseResponse(response: HTTPResponse) обробляє отриману відповідь.

2.10. HTTPResponse:

Опис: Клас, що представляє відповідь сервера на HTTP запит.

Атрибути:

- statusCode: Integer код статусу HTTP (наприклад, 200, 404, 301).

- headers: Map<String, String> заголовки відповіді (для зберігання метаданих).
- body: String тіло відповіді

Методи:

- getStatusCode(): Integer повертає код статусу HTTP.
- getBody(): String повертає вміст тіла відповіді.
- handleRedirect() обробляє редиректи.
- handleError() обробляє помилки

3. Вибрати 3 варіанти використання та написати за ними сценарії використання.

3.1. Користувач – Введення URL:

Передумови: Користувач відкрив веб-браузер, має доступ до Інтернету та бажає перейти на конкретний сайт.

Постумови: Браузер спробує підключитись до введеного URL, передавши запит на сервер, і почне завантажувати веб-сторінку.

Сторони взаємодії: Користувач, Веб-браузер.

Короткий опис: Користувач вводить адресу веб-сайту в адресний рядок браузера, після чого браузер ініціює HTTP-запит до серверу.

Основний перебіг подій:

- Користувач відкриває браузер.
- Користувач вводить URL у адресний рядок.
- Браузер обробляє введену адресу і здійснює HTTP-запит до веб-сервера для завантаження сторінки.

Винятки: Користувач вводить неправильний або неповний URL або неіснуючу адресу сайту.

Примітки: у разі відсутності протоколу, браузер автоматично його додає.

3.2. Веб-сервер - Завантаження сторінки.

Передумови: Браузер надіслав запит на сервер з адресою веб-сторінки, яку потрібно завантажити. Сервер налаштований на обробку запитів і відповідає на них.

Постумови: Веб-сервер відповідає браузеру з HTML-документом та іншими необхідними ресурсами (зображення, CSS, JavaScript файли тощо).

Сторони взаємодії: Користувач, Веб-сервер.

Короткий опис: Після отримання запиту на певну URL-адресу веб-сервер генерує відповідь, що містить HTML-код та інші ресурси для відображення сторінки у браузері.

Основний перебіг подій:

- Браузер надсилає HTTP-запит на сервер для завантаження сторінки.
- Веб-сервер обробляє запит і формує HTTP-відповідь, що містить HTML-документ та інші необхідні ресурси.
- Сервер надсилає HTTP-відповідь назад до браузера.

Винятки: Проблеми із сервером чи пошкоджені файли-ресурси.

Примітки: відсутні.

3.3. Браузер - Обробка HTTP відповіді.

Передумови: Браузер отримав відповідь від сервера у вигляді HTTP статусу та вмісту (HTML, CSS, зображення тощо).

Постумови: Браузер інтерпретує HTTP відповідь та відповідно відображає контент на екран. Якщо є помилка (наприклад, 404), браузер відображає відповідну сторінку помилки.

Сторони взаємодії: Браузер, Веб-сервер.

Короткий опис: Після отримання відповіді від сервера, браузер аналізує HTTP статус, інтерпретує його та відображає контент.

Якщо відповідь містить помилку (наприклад, 404), браузер відображає сторінку з помилкою.

Основний перебіг подій:

- Браузер отримує HTTP-відповідь від сервера і перевіряє код статусу відповіді.
- Якщо відповідь успішна, браузер відображає сторінку з ресурсами.
- Якщо ні, то відображається сторінка з помилкою.

Винятки: Відповідь з помилкою або пошкодженим контентом.

Примітки: відсутні.

4. 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.

4.1. Java-коди для класів:

```
public class AddressBar {
    private String url;
    private String protocol;

    public String getURL() {
        return url;
    }

    public Boolean validateURL(String url) {
        return true;
    }
}
public class Browser {
    private AddressBar addressBar;
    private WebPage currentPage;

    public void loadPage(String url) {

    }

    public void displayPage(WebPage page) {

    }

    public void handleError(Integer code) {

    }

}
public class CSSFile {
    private String fileName;
    private String filePath;
```

```
    private String content;

    public void loadCSS() {

    }

}
public class HTMLFile {
    private String fileName;
    private String filePath;
    private String content;

    public void loadHTML() {

    }

}
public class HTTPRequest {
    private String url;
    private String method;

    public HTTPResponse sendRequest() {

    }

    public void parseResponse(HTTPResponse response) {

    }
}
import java.util.Map;

public class HTTPResponse {
    private Integer statusCode;
    private Map<String, String> headers;
    private String body;

    public Integer getStatusCode() {
        return statusCode;
    }

    public String getBody() {
        return body;
    }

    public void handleRedirect() {

    }

    public void handleError() {

    }
}
public class ImageFile {
    private String fileName;
    private String filePath;
    private String content;

    public void loadImage() {

    }

}
public class JSFile {
    private String fileName;
    private String filePath;
    private String content;
```

```
    public void loadJS() {

    }

}
public class WebPage {
    private HTMLFile htmlResources;
    private CSSFile cssResources;
    private JSFile jsResources;
    private ImageFile imageResources;

    public void parseHTML() {

    }

    public void loadResources() {

    }

}
import java.util.List;

public class WebServer {
    private String host;
    private List<String> resources;
    private List<WebPage> pages;

    public HTTPResponse processRequest(HTTPRequest request) {

    }

    public void sendResponse(HTTPResponse response) {

    }

}
```

4.2. Структура БД:

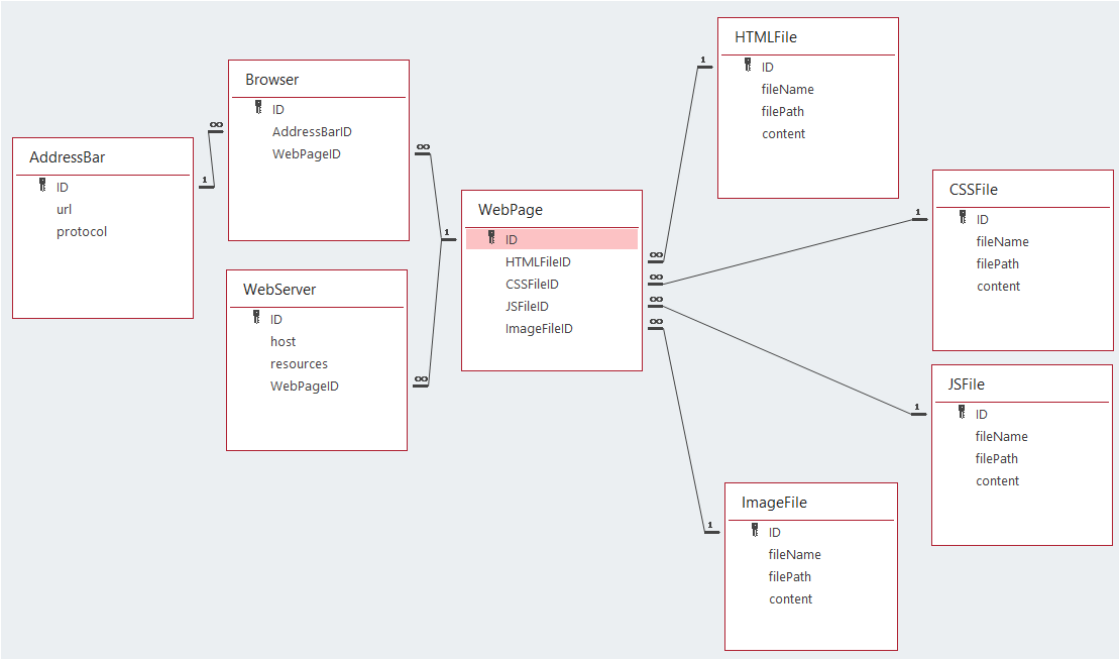

Рис.2 – Структура БД системи

5. Нарисувати діаграму класів для реалізованої частини системи.

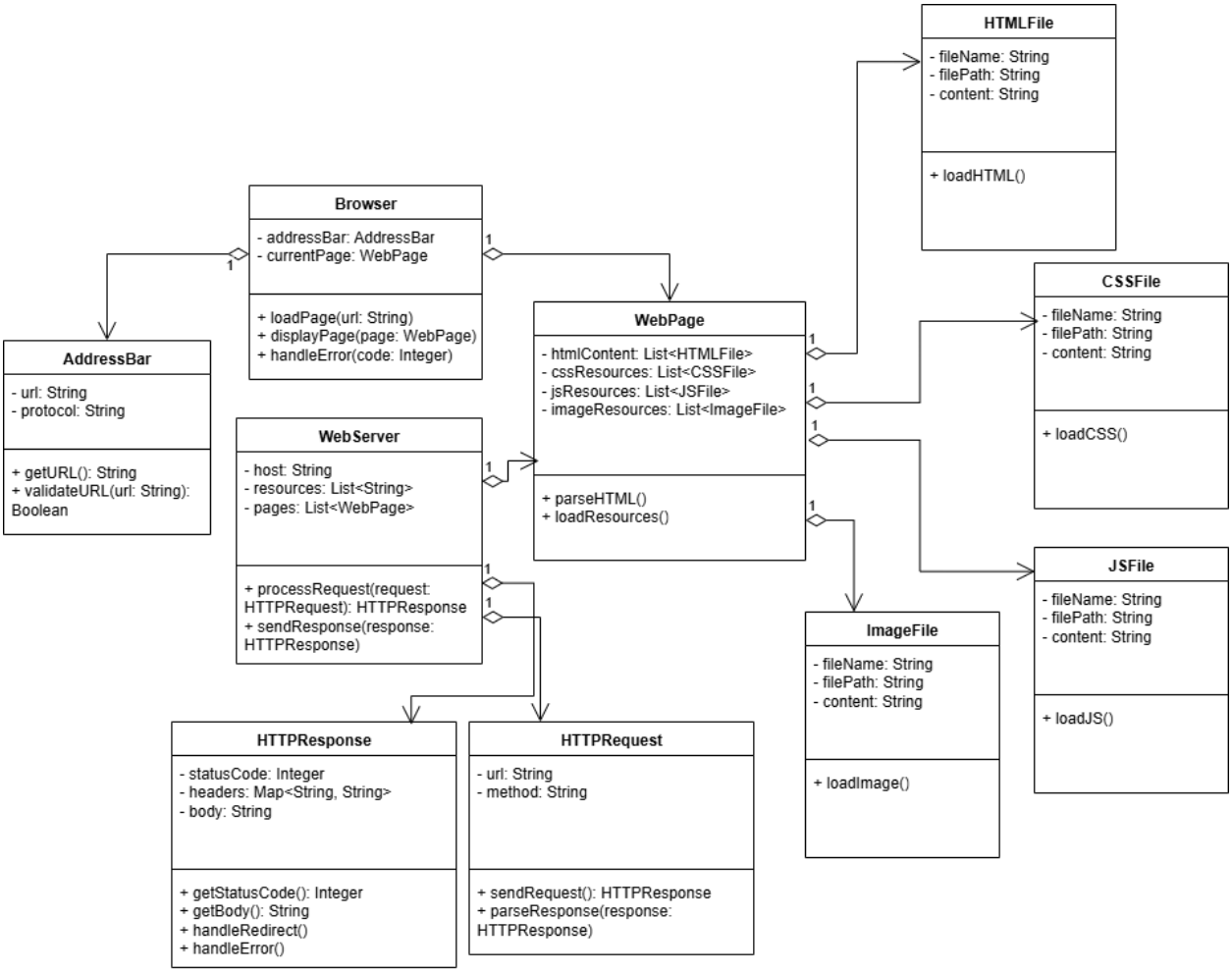

Рис. 3 – Діаграма класів системи

Висновок: на даному лабораторному занятті я познайомився з мовою UML та які інструменти вона пропонує для побудови діаграм. Також я дізнався про основні елементи діаграм послідовності і класів, навчився їх проєктувати та рисувати. Також я відпрацював навички написання коду по шаблону Repository, а також робити загальну структуру БД в середовищі MS Access.

Відповіді на контрольні питання:

1. Що таке UML?

   Мова UML є загальноцільовою мовою 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 в UML – це діаграма, де показуються класи, інтерфейси та відносини між ними.

3. Які діаграми UML називають канонічними?
   - 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).

4. Що таке діаграма варіантів використання?

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

5. Що таке варіант використання?

   Варіант використання - 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?
   - **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. Позначається стрілкою.

- 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.

7. Що таке сценарій?

   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.

8. Що таке діаграма класів?

   Діаграми класів - одна із форм статичного опису системи з погляду її проєктування, показуючи її структуру.

9. Які зв’язки між класами ви знаєте?

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

10. Чим відрізняється композиція від агрегації?

    В композиції компоненти не можуть існувати окремо один від одного, на відміну від агрееації.

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

    Агрегація має порожній ромб, композиція заповнений.

12. Що являють собою нормальні форми баз 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.

13. Що таке фізична модель бази даних? Логічна?

    Фізична модель бази даних описує, як дані будуть зберігатися на фізичному рівні. Фактично, типи даних.

    Логічна модель описує зв’язки між таблицями.

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

    По суті, ми створюємо БД на основі програмних класів, оскільки беремо звідти їх тип даних, і зв’язки.