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 «Технології розробки програмного забезпечення»

Основи проектування

Виконав:
студент групи ІА-34
Швець Роман Вадимович

Перевірив:
асистент кафедри ІСТ
Мягкий Михайло Юрійович

Київ - 2025

Тема: Основи програмування.

Мета: Обрати зручну систему побудови UML-діаграм і навчитися будувати діаграми варіантів використання для системи, яку ми проектуємо, розробляти сценарії варіантів використання і будувати діаграми класів предметної області.

Теоретичні відомості

UML – це уніфікована графічна мова, призначена для опису, моделювання та документування програмних систем, бізнес-процесів і складних об’єктів. Вона поєднує напрацювання різних підходів у програмній інженерії, тому використовується як для концептуального, так і для логічного та фізичного рівнів моделювання. У процесі об’єктно-орієнтованого аналізу та проектування UML забезпечує поступовий перехід від загальних схем до детальних моделей, послідовно доповнюючи їх деталями та відображаючи різні аспекти майбутньої системи. Моделі можна розглядати на різних рівнях абстракції – від концептуального (загальний опис) до логічного (структура та взаємодії) й фізичного (реальні компоненти та середовище функціонування).

Усі ці уявлення фіксуються за допомогою діаграм. Діаграма – це графічне подання елементів моделі та їхніх зв’язків у вигляді графа з визначеною семантикою вузлів і ребер. Стандарт UML включає кілька основних типів діаграм: варіантів використання (use-case), класів, кооперації, послідовності, станів, діяльності, компонентів і розгортання. У сукупності вони відображають повну картину роботи системи – від зовнішніх вимог і сценаріїв до внутрішньої структури та фізичного розташування компонентів.

Діаграма варіантів використання є початковою концептуальною моделлю, яка визначає межі функціональності системи, основні сценарії та вимоги до неї. Вона створюється на етапі збору та аналізу вимог, коли аналітики й розробники з’ясовують, які функції має забезпечувати система. Така діаграма не розкриває внутрішню будову, а відображає лише взаємодію зовнішніх користувачів або систем (акторів) із системою.

Актори – це зовнішні суб’єкти, що взаємодіють із моделлю: люди, пристрої, інші програми або системи. Зазвичай їх розглядають як ролі (наприклад,

«покупець», «касир», «адміністратор»). Варіанти використання (use cases) описують послуги чи сценарії, які система надає актору, тобто послідовність дій у процесі їх взаємодії (наприклад, реєстрація, авторизація, оформлення замовлення, перевірка рахунку). У діаграмі вони зображуються еліпсами з короткими назвами.

Зв’язки між акторами та варіантами використання демонструють тип взаємодії.

Асоціація – базовий зв’язок між актором і варіантом використання, що показує, якою функцією користується актор. Вона може бути без напрямку (коли ініціатор взаємодії не визначений) або з напрямком (якщо відомо, хто ініціює дію).

Узагальнення – відображає спадкування властивостей чи поведінки елементів одного типу. Наприклад, актор «Адміністратор» успадковує властивості «Користувача» та має додаткові. Аналогічно у варіантах використання: «Оплата карткою» є уточненням варіанта «Оплата замовлення».

Залежність – показує, що зміни одного елемента впливають на інший. Серед її підтипів:

«Включення» (include) – коли один варіант використання завжди містить у собі інший (наприклад, оформлення замовлення включає автентифікацію користувача).

«Розширення» (extend) – коли основний сценарій за певних умов може бути доповнений додатковими діями.

Діаграми варіантів використання та їхні зв’язки формують цілісне бачення системи на концептуальному рівні. Вони слугують основою для побудови діаграм класів, послідовностей, компонентів та інших, дозволяючи ще до початку проєктування й реалізації отримати узгоджену картину функцій, поведінки та структури складних систем.

Хід роботи

Темою нашого проекту та подальших лабораторних робіт стане створення графічного редактора.

Графічний редактор повинен вміти створювати / редагувати растрові (або векторні) зображення в 2-3 основних популярних форматах (bmp, png, jpg), мати

панель інструментів для створення графічних примітивів, вибору кольорів, нанесення тексту, додавання найпростіших візуальних ефектів (ч/б растр, інфрачервоний растр і т. п.), роботи з шарами.

Спроектуємо діаграму варіантів використання (use-case діаграму) відповідно до нашої теми:

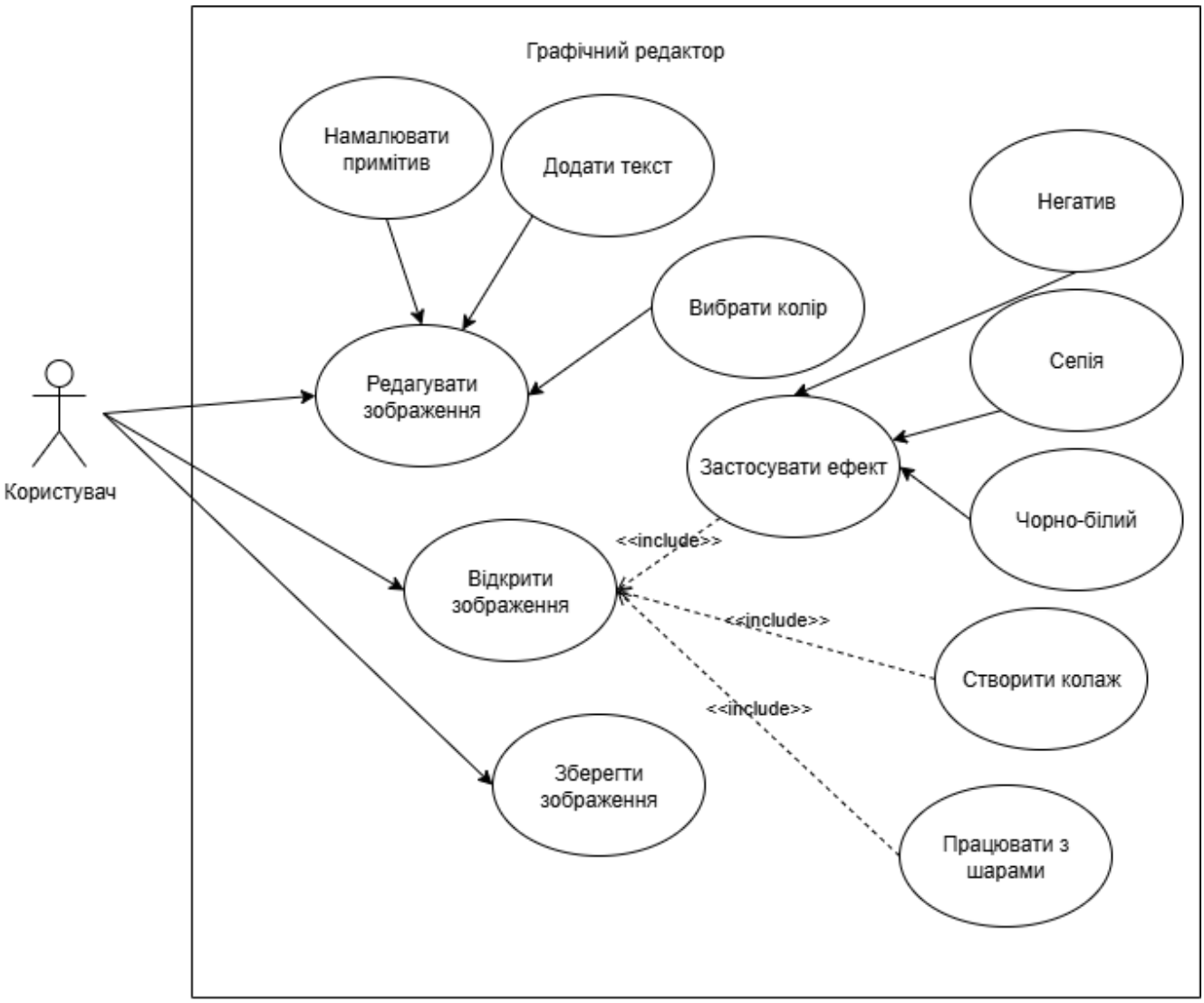


Рис. 1 – Діаграма варіантів використання

У системі «Графічний редактор» користувач може відкривати та зберігати зображення, створювати нові файли та редагувати їх. Редагування включає роботу з графічними примітивами, текстом, кольорами, а також із шарами та створенням колажів. Крім того, користувач може застосовувати прості візуальні ефекти (чорно-білий, сепія, негатив).

Далі спроектуємо діаграму класів нашої предметної області:

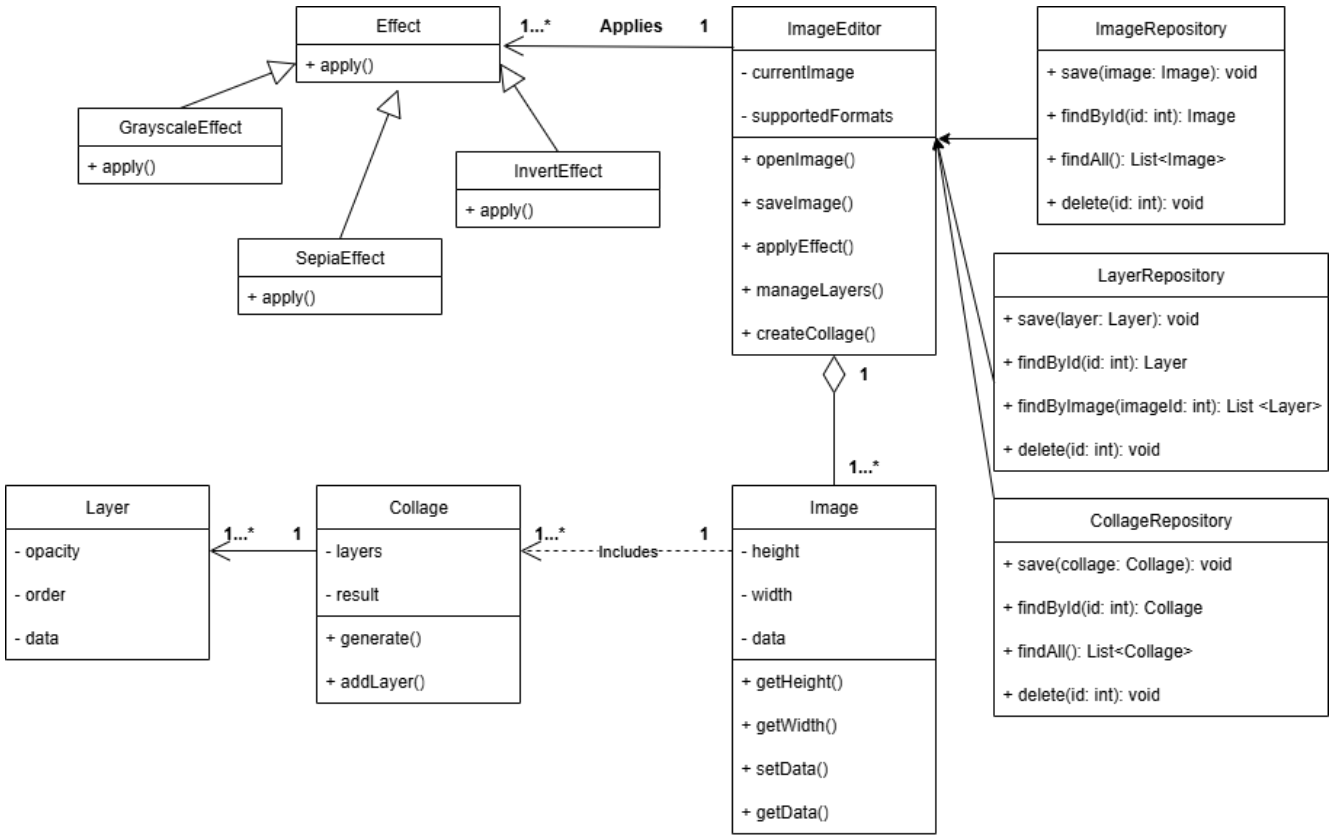


Рис. 2 – Діаграма класів

Система для редагування зображень використовує Effect з методом apply() та нащадками GrayscaleEffect, SepiaEffect, InvertEffect. ImageEditor керує currentImage, підтримуваними форматами і забезпечує відкриття, збереження, застосування ефектів, управління шарами та створення колажів. Image містить height, width, data, a Layer – opacity, order, data. Collage складається з шарів і зображень з методами generate() та addLayer(). Репозиторії (ImageRepository, LayerRepository, CollageRepository) зберігають та шукають об’єкти. ImageEditor працює з Image, Layer і Collage, ефекти застосовуються через нього. Effect – фільтри, ImageEditor – керування, Image/Layer/Collage — об’єкти, репозиторії – сховище.

Зв’язки:

ImageEditor – Image (агрегація).

ImageEditor – Effect (асоціація).

Collage – Image (асоціація 1..).

Collage – Layer (композиція 1..).

Effect –підкласи (успадкування).

Тепер оберемо 3 варіанти використання і опишемо їх:

Варіант використання 1: «Застосувати ефект до зображення»

Передумови: Відкрите зображення у редакторі.

Постумови:

Якщо виконано успішно — система відображає зображення з вибраним ефектом (Grayscale, Sepia або Invert).

Якщо сталася помилка — зображення залишається без змін.

Взаємодіючі сторони: Користувач, Редактор зображень.

Короткий опис: Користувач застосовує один із доступних ефектів до відкритого зображення.

Основний потік подій:

1. Користувач відкриває зображення.
2. У меню вибирає потрібний ефект.
3. Система застосовує ефект до поточного зображення.
4. У вікні попереднього перегляду відображається результат.

Винятки:

Обрано ефект, несумісний із поточним форматом зображення → повідомлення про помилку.

Варіант використання 2: «Керування шарами»

Передумови: Відкрите зображення або створюється новий проект.

Постумови:

Якщо операція успішна — додано, змінено чи видалено шар.

Якщо сталася помилка — зміни не застосовуються.

Взаємодіючі сторони: Користувач, Редактор зображень.

Короткий опис: Користувач працює з шарами зображення (додає нові, змінює порядок, видаляє непотрібні).

Основний потік подій:

1. Користувач обирає команду «Додати шар».
2. Система створює новий шар і додає його до колекції шарів.
3. Користувач редагує шар: змінює позицію, прозорість чи застосовує ефект.
4. У разі потреби шар можна видалити.
5. Система відображає оновлений результат.

Винятки:

Перевищено допустиму кількість шарів → повідомлення про помилку.

Варіант використання 3: «Створити колаж»

Передумови: Є кілька відкритих зображень або доданих шарів.

Постумови:

Якщо виконано успішно — створено колаж з обраних зображень/шарів.

Якщо сталася помилка — колаж не створюється.

Взаємодіючі сторони: Користувач, Редактор зображень.

Короткий опис: Користувач створює колаж, комбінуючи кілька шарів або зображень.

Основний потік подій:

1. Користувач запускає команду «Створити колаж».
2. Система пропонує вибрати зображення/шари для колажу.
3. Користувач додає потрібні елементи.
4. Система генерує колаж.
5. У вікні попереднього перегляду відображається результат.

Винятки:

Вибрано занадто мало зображень (наприклад, одне) → повідомлення про помилку.

Вихідні коди класів системи

Лістинг 1 – клас Image

```
import jakarta.persistence.*;

@Entity
@Table(name = "images")
public class Image {

    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "file_name", nullable = false)
    private String fileName;

    @Column(nullable = false)
    private String format; // SVG, EPS

    @Lob
    @Column(nullable = false)
    private byte[] data;

    public Image() {}

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getFileName() { return fileName; }
    public void setFileName(String fileName) { this.fileName = fileName; }

    public String getFormat() { return format; }
    public void setFormat(String format) { this.format = format; }

    public byte[] getData() { return data; }
    public void setData(byte[] data) { this.data = data; }
}
```

Лістинг 2 – абстрактний клас Effect

```
import jakarta.persistence.*;

@Entity
@Inheritance(strategy = InheritanceType.JOINED)
@Table(name = "effects")
public abstract class Effect {

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

    @Column(nullable = false)
    private String name;

    public Effect() {}

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
}
```

Лістинг 3 – клас GrayscaleEffect

```
import jakarta.persistence.*;

@Entity
@Table(name = "grayscale_effects")
public class GrayscaleEffect extends Effect {

    @Column(nullable = false)
    private boolean applied;
```

```
    public GrayscaleEffect() {}

    public boolean isApplied() { return applied; }
    public void setApplied(boolean applied) { this.applied = applied; }
}
```

Лістинг 4 – клас SepiaEffect

```
import jakarta.persistence.*;

@Entity
@Table(name = "sepia_effects")
public class SepiaEffect extends Effect {

    @Column(nullable = false)
    private boolean applied;

    public SepiaEffect() {}

    public boolean isApplied() { return applied; }
    public void setApplied(boolean applied) { this.applied = applied; }
}
```

Лістинг 5 – клас InvertEffect

```
import jakarta.persistence.*;

@Entity
@Table(name = "invert_effects")
public class InvertEffect extends Effect {

    @Column(nullable = false)
    private boolean applied;

    public InvertEffect() {}

    public boolean isApplied() { return applied; }
    public void setApplied(boolean applied) { this.applied = applied; }
}
```

Лістинг 6 – клас Layer

```
import jakarta.persistence.*;

@Entity
@Table(name = "layers")
public class Layer {

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

    @ManyToOne
    @JoinColumn(name = "image_id", nullable = false)
    private Image image;

    @Column(name = "layer_order")
    private int layerOrder;

    public Layer() {}

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
```

```
    public Image getImage() { return image; }
    public void setImage(Image image) { this.image = image; }

    public int getLayerOrder() { return layerOrder; }
    public void setLayerOrder(int layerOrder) { this.layerOrder = layerOrder; }
}
```

Лістинг 7 – клас Collage

```
import jakarta.persistence.*;
import java.util.ArrayList;
import java.util.List;

@Entity
@Table(name = "collages")
public class Collage {

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

    @Column(nullable = false)
    private String title;

    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)
    @JoinColumn(name = "collage_id")
    private List<Layer> layers = new ArrayList<>();

    public Collage() {}

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    public List<Layer> getLayers() { return layers; }
    public void setLayers(List<Layer> layers) { this.layers = layers; }

    public void addLayer(Layer layer) {
        layers.add(layer);
    }

    public void removeLayer(Layer layer) {
        layers.remove(layer);
    }
}
```

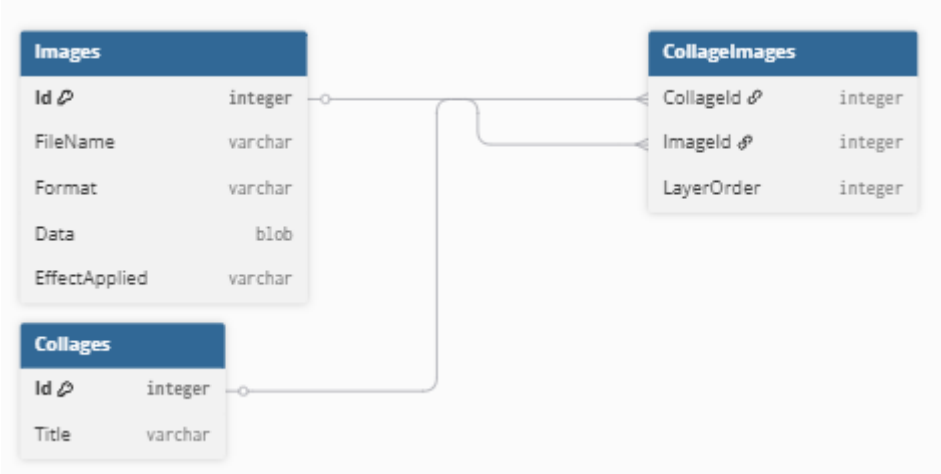
Лістинг 8 – клас models.sql

```
CREATE TABLE Images (
    Id INTEGER PRIMARY KEY,
    FileName VARCHAR(255) NOT NULL,
    Format VARCHAR(50) NOT NULL,
    Data BLOB,
    EffectApplied VARCHAR(50)
);

CREATE TABLE Collages (
    Id INTEGER PRIMARY KEY,
    Title VARCHAR(255) NOT NULL
);
```

```
CREATE TABLE CollageImages (
    CollageId INTEGER NOT NULL,
    ImageId INTEGER NOT NULL,
    LayerOrder INTEGER,
    PRIMARY KEY (CollageId, ImageId),
    FOREIGN KEY (CollageId) REFERENCES Collages(Id),
    FOREIGN KEY (ImageId) REFERENCES Images(Id)
);
```

Діаграма структури бази даних



Висновок:

У ході виконання роботи ми проаналізували предметну область та визначили основні сутності графічного редактора для роботи з векторними зображеннями у форматах SVG та EPS. Побудували діаграму варіантів використання, яка відображає ключові функції системи (завантаження та збереження зображень, застосування ефектів, створення колажів). Створили діаграму класів, що описує взаємозв’язки між зображеннями, ефектами, колажами та шарами.

Також розробили SQL-модель даних, це забезпечує наочність та узгодженість між логічною моделлю та її реалізацією в СУБД.

Отже, результатом роботи стала побудова цілісної моделі майбутнього застосунку, яка закладає основу для подальшого проектування та реалізації функціоналу редактора.

Відповіді на контрольні запитання:

1. Що таке UML?

UML (Unified Modeling Language) – уніфікована мова моделювання, яка використовується для візуалізації, опису та документування систем, зокрема програмних.

2. Що таке діаграма класів UML?

Діаграма класів показує структуру системи: класи, їх атрибути, методи та зв’язки між ними.

3. Які діаграми UML називають канонічними?

До канонічних діаграм UML належать:

- діаграма класів,
- діаграма варіантів використання,
- діаграма послідовності,
- діаграма станів,
- діаграма компонентів,
- діаграма розгортання.

4. Що таке діаграма варіантів використання?

Це діаграма, яка відображає взаємодію користувачів (акторів) із системою через варіанти використання (use cases).

5. Що таке варіант використання?

Use case – це опис конкретної функції системи, яку може виконати користувач або інша система.

6. Які відношення можуть бути відображені на діаграмі використання?

- асоціація (actor – use case),
- include («включає»),
- extend («розширює»),
- generalization (успадкування акторів або варіантів).

7. Що таке сценарій?

Сценарій – це послідовність дій (кроків) взаємодії користувача із системою для реалізації конкретного use case.

8. Що таке діаграма класів?

Діаграма, що описує класи, їх атрибути, методи та зв’язки (асоціації, композиції, агрегації, наслідування).

9. Які зв’язки між класами ви знаєте?

Асоціація, залежність, агрегація, композиція, наслідування, реалізація.

10. Чим відрізняється композиція від агрегації?

Агрегація – «слабке ціле» (частини можуть існувати окремо), композиція – «сильне ціле» (частини не існують без цілого).

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

Агрегація позначається порожнім ромбом на стороні цілого, композиція позначається зафарбованим ромбом.

12. Що являють собою нормальні форми баз даних?

Це правила організації таблиць, що зменшують надлишковість і аномалії даних (1НФ, 2НФ, 3НФ, БКНФ тощо).

13. Що таке фізична модель бази даних? Логічна?

Логічна модель описує сутності, атрибути, зв’язки без урахування конкретної СУБД, фізична враховує специфіку СУБД (типи даних, індекси, ключі, структуру таблиць).

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

Таблиця БД відповідає класу, рядки таблиці – об’єктам (екземплярам класу), а стовпці – атрибутам класу.