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 розробки програмного забезпечення

“ Основи проектування”

Виконав студент групи ІА–33:
Ященко К.А.

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

Зміст

Завдання:

1. Ознайомитись з короткими теоретичними відомостями щодо UML та процесу проєктування програмних систем.
2. Проаналізувати обрану тему лабораторної роботи та спроєктувати діаграму варіантів використання відповідно до предметної області.
3. Спроєктувати діаграму класів предметної області.
4. Вибрати три варіанти використання та розробити для них детальні сценарії використання.
5. На основі спроєктованої діаграми класів предметної області розробити основні класи системи та структуру бази даних. Класи доступу до

даних повинні реалізовувати шаблон Repository для взаємодії з базою даних.
6. Побудувати діаграму класів для реалізованої частини системи.
7. Підготувати звіт щодо виконання лабораторної роботи, який повинен містити:

    - діаграму варіантів використання;
    - діаграму класів системи;
    - вихідні коди класів системи;
    - зображення структури бази даних.

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 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) — це 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 Case Diagram) — це UML-діаграма, за допомогою якої у графічному вигляді можна зобразити вимоги до системи, що розробляється. Вона є вихідною концептуальною моделлю проєктованої системи та не описує її внутрішню побудову.

Діаграми варіантів використання є відправною точкою при зборі вимог до програмного продукту та його реалізації. Вони дозволяють отримати більш повне уявлення про функціональні можливості системи та документувати їх.

Сценарії використання — це текстові уявлення процесів взаємодії користувача з системою. Вони описують покрокову послідовність дій, що призводить до досягнення конкретної мети, та однозначно визначають кінцевий результат.

Діаграми класів використовуються для статичного опису програмної системи з точки зору її структури. На таких діаграмах зображуються класи, їх атрибути, методи та зв’язки між ними.

Клас є основним будівельним елементом програмної системи. Він має назву, набір атрибутів та операцій. UML-класи мають пряме відображення у класах мов програмування, що дозволяє використовувати їх для генерації програмного коду.

Логічна модель бази даних являє собою структуру таблиць, зв’язків та інших логічних елементів, необхідних для зберігання та обробки даних. Проєктування бази даних виконується у тісному зв’язку з архітектурою програмної системи.

Тема

26. Download Manager

(Iterator, Command, Observer, Template Method, Composite, P2P)

Менеджер завантажень призначений для скачування файлів з мережі Інтернет за протоколами HTTP та HTTPS. Система повинна підтримувати можливість відновлення завантаження з місця зупинки, розподіл швидкості між активними завантаженнями, а також ведення статистики завантажень.

Програмний продукт повинен забезпечувати інтеграцію з основними веббраузерами, такими як Firefox, Opera, Internet Explorer та Google Chrome. Архітектура системи має базуватися на використанні шаблонів проєктування, що забезпечують гнучкість, розширюваність та зручність супроводу програмного забезпечення.

Хід роботи

Діаграма варіантів використання

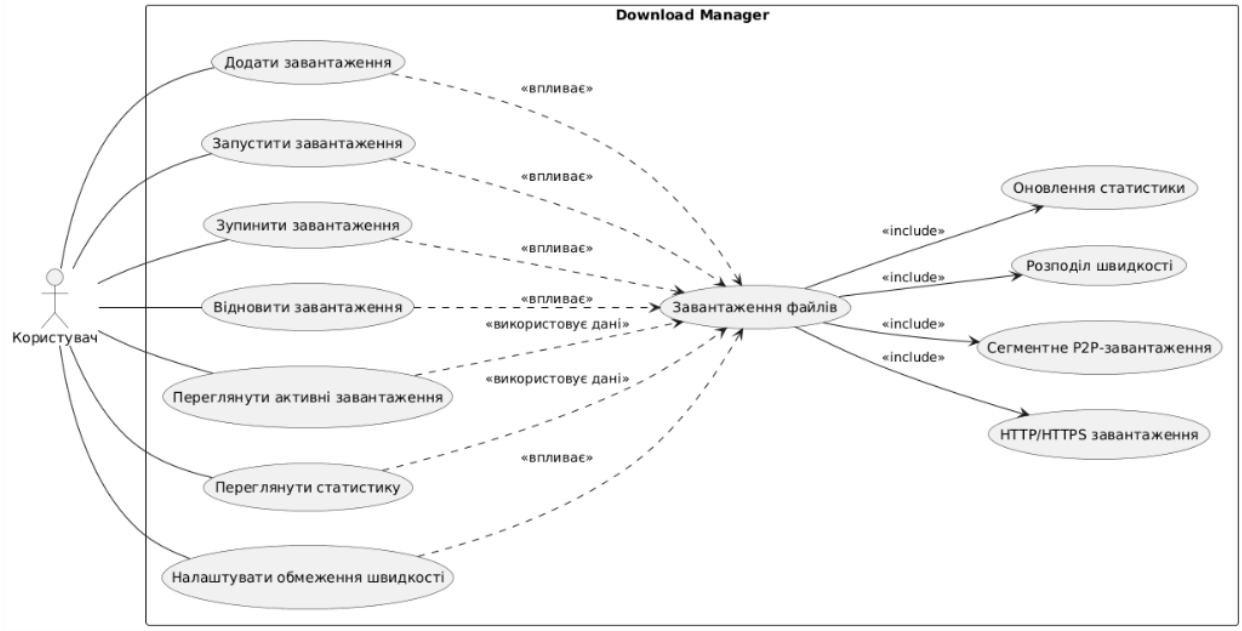


Рис. 1 – Діаграма варіантів використання

Сценарії використання

Сценарій 1. Додавання та запуск нового завантаження

Передумови:

- Download Manager встановлений і запущений.
- Користувач має коректне HTTP або HTTPS посилання на файл.
- У системі достатньо місця на диску для збереження файлу.

Постумови:

- Нове завантаження створене у системі.
- Файл починає завантажуватись у сегментному режимі.
- Початковий стан завантаження збережено у локальній базі даних.

Взаємодіючі сторони:

- Користувач
- Download Manager (десктопний додаток)

Короткий опис:

Користувач додає URL файлу та запускає процес завантаження, який відбувається з використанням сегментного P2P-підходу через HTTP/HTTPS.

Основний перебіг подій:

1. Користувач відкриває головне вікно Download Manager.
2. Користувач вводить URL файлу у відповідне поле.
3. Користувач натискає кнопку «Додати завантаження».
4. Система створює нову задачу завантаження.
5. Система розбиває файл на сегменти.
6. Запускається сегментне P2P-завантаження через HTTP/HTTPS.
7. Система починає відображати прогрес завантаження.

Винятки:

- Якщо URL недоступний або некоректний — система повідомляє про помилку.
- Якщо неможливо встановити HTTP-з’єднання — завантаження не запускається.

Примітки:

- Даний сценарій ініціює центральний процес «Завантаження файлів».
- Реалізує патерни Command, Composite, Template Method.

Сценарій 2. Призупинення та відновлення завантаження

Передумови:

- Завантаження файлу вже запущене.
- Частина файлу вже завантажена.

Постумови:

- Стан завантаження збережений у базі даних.
- Після відновлення завантаження продовжується з місця зупинки.

Взаємодіючі сторони:

- Користувач
- Download Manager

Короткий опис:

Користувач тимчасово призупиняє активне завантаження та згодом відновлює його без втрати вже завантажених даних.

Основний перебіг подій:

1. Користувач обирає активне завантаження зі списку.
2. Користувач натискає кнопку «Зупинити».

3. Система зупиняє всі активні сегменти.
4. Поточний стан сегментів зберігається у SQLite.
5. Користувач натискає кнопку «Відновити завантаження».
6. Система зчитує збережений стан.
7. Завантаження продовжується з попередньої позиції.

Винятки:

- Якщо дані стану пошкоджені — система повідомляє про неможливість відновлення.
- Якщо файл на сервері більше недоступний — завантаження завершується з помилкою.

Примітки:

- Сценарій демонструє підтримку resume-завантаження.
- Реалізує патерни Command, Iterator, Composite.

Сценарій 3. Перегляд статистики завантажень

Передумови:

- У системі вже виконувались завантаження.
- Статистичні дані збережені у локальній базі даних.

Постумови:

- Користувач отримує статистичну інформацію про завантаження.

Взаємодіючі сторони:

- Користувач
- Download Manager

Короткий опис:

Користувач переглядає накопичену статистику щодо завантажень, включаючи швидкість, обсяг даних та кількість активних сегментів.

Основний перебіг подій:

1. Користувач відкриває розділ «Статистика».
2. Система звертається до локальної бази даних.
3. Дані про завантаження агрегуються.
4. Статистика відображається у графічному або табличному вигляді.

Винятки:

- Якщо статистичні дані відсутні — система повідомляє користувача.
- Якщо база даних недоступна — відображається повідомлення про помилку.

Примітки:

- Сценарій використовує дані, зібрані Observer-модулями.
- Реалізує патерн Observer.

Діаграма класів

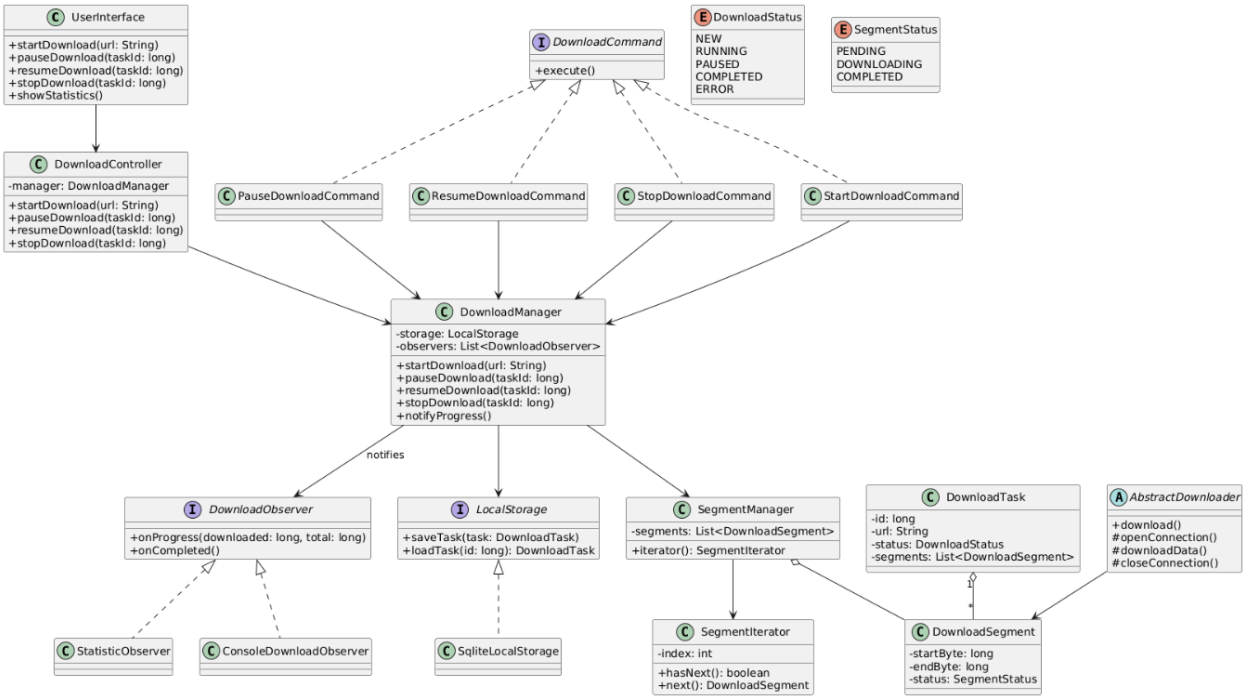


Рис. 2 – Діаграма класів

1. UserInterface – DownloadController

- Тип відносин: Асоціація (1 → 1).
- Пояснення:
  Клас UserInterface має посилання на DownloadController і використовується для передачі дій користувача (додавання, запуск, пауза, відновлення та зупинка завантаження). Користувацький інтерфейс не містить бізнес-логіки, а лише делегує її контролеру.

2. DownloadController – DownloadManager

- Тип відносин: Асоціація (1 → 1).
- Пояснення:
  DownloadController керує життєвим циклом завантажень і делегує всі основні операції (startDownload(), pauseDownload(), resumeDownload(),

stopDownload()) класу DownloadManager, який реалізує основну логіку системи.

3. DownloadController – DownloadCommand

- Тип відносин: Асоціація (1 → *).
- Пояснення:
  Контролер створює та виконує об’єкти команд (StartDownloadCommand, PauseDownloadCommand, ResumeDownloadCommand, StopDownloadCommand), що реалізують шаблон Command. Це дозволяє інкапсулювати дії користувача у вигляді окремих об’єктів.

4. DownloadCommand – DownloadManager

- Тип відносин: Асоціація (1 → 1).
- Пояснення:
  Кожна команда містить посилання на DownloadManager і викликає відповідний метод менеджера під час виконання команди (execute()).

5. DownloadManager – DownloadTask

- Тип відносин: Агрегація (1 → *).
- Пояснення:
  DownloadManager керує множиною об’єктів DownloadTask. Завдання можуть існувати незалежно, але їх життєвий цикл координується менеджером завантажень.

6. DownloadTask – DownloadSegment

- Тип відносин: Композиція (1 → *).
- Пояснення:
  DownloadTask складається з набору сегментів (DownloadSegment), які представляють частини файлу. Сегменти не можуть існувати без відповідного завдання завантаження, що відповідає шаблону Composite.

7. DownloadManager – SegmentManager

- Тип відносин: Асоціація (1 → 1).

- Пояснення:
  DownloadManager використовує SegmentManager для управління сегментами файлу та організації ітерації над ними під час процесу завантаження.

8. SegmentManager – SegmentIterator

- Тип відносин: Асоціація (1 → 1).
- Пояснення:
  SegmentManager створює та повертає об’єкт SegmentIterator, який реалізує шаблон Iterator для послідовного доступу до сегментів без розкриття внутрішньої структури колекції.

9. DownloadManager – DownloadObserver

- Тип відносин: Асоціація (1 → *).
- Пояснення:
  DownloadManager підтримує список спостерігачів (DownloadObserver) і повідомляє їх про зміни стану завантаження (прогрес, завершення). Це реалізація шаблону Observer.

10. DownloadObserver – ConsoleDownloadObserver / StatisticObserver

- Тип відносин: Наслідування.
- Пояснення:
  Класи ConsoleDownloadObserver та StatisticObserver реалізують інтерфейс DownloadObserver, надаючи різні способи обробки подій завантаження (вивід у консоль, збір статистики).

11. DownloadManager – AbstractDownloader

- Тип відносин: Асоціація (1 → 1).
- Пояснення:
  DownloadManager використовує абстрактний клас AbstractDownloader, який реалізує шаблон Template Method та визначає загальний алгоритм завантаження файлу з можливістю перевизначення окремих кроків.

12. DownloadManager – LocalStorage

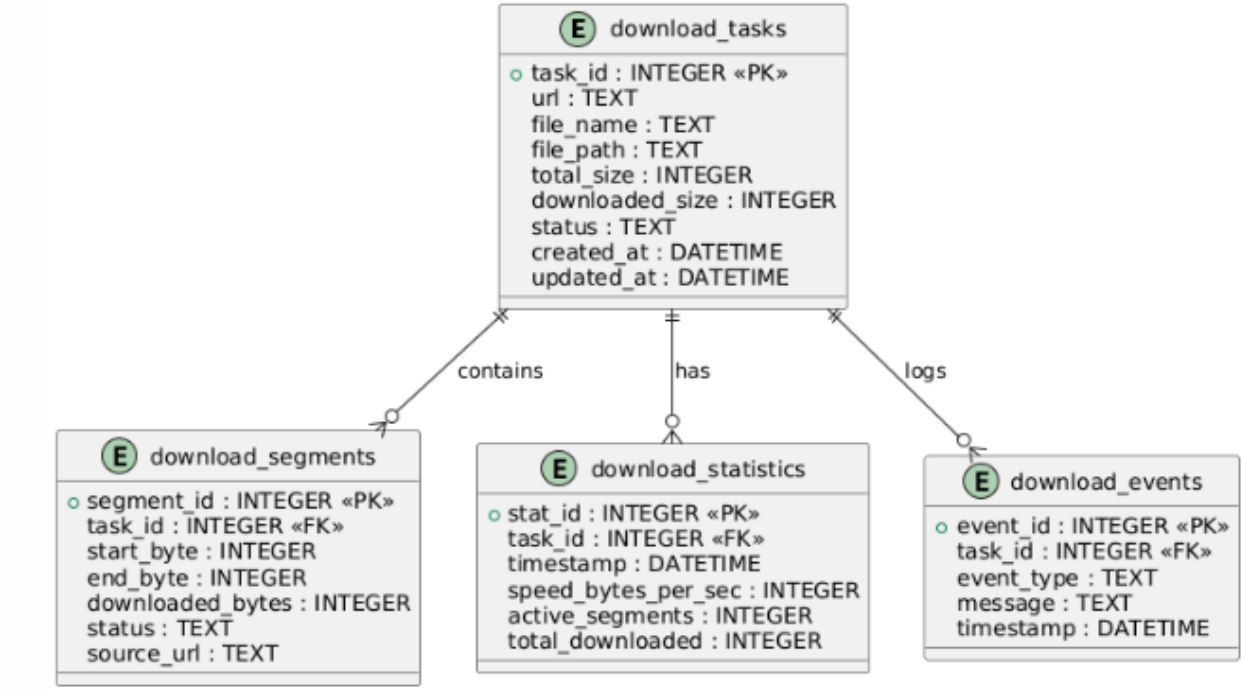
- Тип відносин: Асоціація (1 → 1).

- Пояснення:
  DownloadManager взаємодіє з інтерфейсом LocalStorage для збереження та відновлення інформації про завантаження. Конкретна реалізація (SqliteLocalStorage) інкапсулює роботу з локальною базою даних SQLite.

13. LocalStorage – SqliteLocalStorage

- Тип відносин: Наслідування (реалізація інтерфейсу).
- Пояснення:
  Клас SqliteLocalStorage реалізує інтерфейс LocalStorage і відповідає за фізичне збереження даних у вбудованій базі даних SQLite.

Проєктування БД



Опис БД

1. Таблиця download_tasks
Призначення

Зберігає інформацію про кожне завдання завантаження файлу як цілісну логічну одиницю. Використовується для керування станом завантаження,

відновлення процесу після перезапуску програми та відображення списку завантажень у користувацькому інтерфейсі.

Поля

- task_id INTEGER — унікальний ідентифікатор завдання (PK)
- url TEXT — адреса файлу для завантаження (HTTP/HTTPS)
- file_name TEXT — ім’я файлу
- file_path TEXT — шлях збереження файлу
- total_size INTEGER — загальний розмір файлу (у байтах)
- downloaded_size INTEGER — кількість вже завантажених байтів
- status TEXT — стан завантаження (NEW, DOWNLOADING, PAUSED, COMPLETED, ERROR)
- created_at DATETIME — дата створення завдання
- updated_at DATETIME — дата останнього оновлення

Зв’язки

- Один-до-багатьох з таблицею download_segments
- Один-до-багатьох з таблицею download_statistics
- Один-до-багатьох з таблицею download_events

2. Таблиця download_segments
Призначення

Зберігає інформацію про сегменти файлу, що використовуються для сегментного (P2P-подібного) завантаження через HTTP Range. Дозволяє реалізувати паралельне завантаження та відновлення окремих частин файлу.

Поля

- segment_id INTEGER — унікальний ідентифікатор сегмента (PK)
- task_id INTEGER — ідентифікатор завдання завантаження (FK)
- start_byte INTEGER — початковий байт сегмента
- end_byte INTEGER — кінцевий байт сегмента
- downloaded_bytes INTEGER — кількість завантажених байтів сегмента
- status TEXT — стан сегмента (PENDING, DOWNLOADING, COMPLETED, ERROR)
- source_url TEXT — HTTP-джерело сегмента

Зв’язки

- Багато-до-одного з таблицею download_tasks

3. Таблиця download_statistics
Призначення

Зберігає статистичні дані процесу завантаження. Використовується для відображення швидкості, прогресу та аналізу продуктивності завантажень.

Поля

- stat_id INTEGER — унікальний ідентифікатор запису (PK)
- task_id INTEGER — ідентифікатор завдання (FK)
- timestamp DATETIME — час фіксації статистики
- speed_bytes_per_sec INTEGER — швидкість завантаження
- active_segments INTEGER — кількість активних сегментів
- total_downloaded INTEGER — загальний обсяг завантажених даних

Зв’язки

- Багато-до-одного з таблицею download_tasks

4. Таблиця download_events
Призначення

Зберігає події життєвого циклу завантаження. Використовується для журналювання дій користувача та системних подій (старт, пауза, відновлення, помилки).

Поля

- event_id INTEGER — унікальний ідентифікатор події (PK)
- task_id INTEGER — ідентифікатор завдання (FK)
- event_type TEXT — тип події (START, PAUSE, RESUME, STOP, ERROR)
- message TEXT — опис події
- timestamp DATETIME — час події

Зв’язки

- Багато-до-одного з таблицею download_tasks

Вихідний код без реалізації на мові Java

package com.example.downloadmanager.core;

```
import com.example.downloadmanager.model.DownloadTask;
import com.example.downloadmanager.observer.DownloadObserver;
import com.example.downloadmanager.storage.LocalStorage;

import java.util.ArrayList;
import java.util.List;

public class DownloadManager {

    private LocalStorage storage;
    private List<DownloadObserver> observers;
    private DownloadTask currentTask;

    public DownloadManager(LocalStorage storage) {
        this.storage = storage;
        this.observers = new ArrayList<>();
    }

    public void startDownload(Long taskId) {
        // знайти задачу
        // ініціалізувати сегменти
        // запустити завантаження
    }

    public void pauseDownload(Long taskId) {
        // призупинити всі активні сегменти
    }

    public void resumeDownload(Long taskId) {
```

```
        // відновити завантаження сегментів
    }

    public void stopDownload(Long taskId) {
        // зупинити завантаження
    }

    public void addObserver(DownloadObserver observer) {
        observers.add(observer);
    }

    protected void notifyProgress(long downloaded, long total) {
        for (DownloadObserver observer : observers) {
            observer.onProgress(downloaded, total);
        }
    }

    protected void notifyCompleted() {
        for (DownloadObserver observer : observers) {
            observer.onCompleted();
        }
    }
}
package com.example.downloadmanager.model;

import java.util.ArrayList;
import java.util.List;

public class DownloadTask {
```

```
    private Long id;
    private String url;
    private String fileName;
    private DownloadStatus status;
    private List<DownloadSegment> segments;

    public DownloadTask(String url, String fileName) {
        this.url = url;
        this.fileName = fileName;
        this.status = DownloadStatus.NEW;
        this.segments = new ArrayList<>();
    }

    public void addSegment(DownloadSegment segment) {
        segments.add(segment);
    }

    public List<DownloadSegment> getSegments() {
        return segments;
    }

    public Long getId() {
        return id;
    }
}
package com.example.downloadmanager.model;

public class DownloadSegment {
```

```
    private long startByte;
    private long endByte;
    private SegmentStatus status;

    public DownloadSegment(long startByte, long endByte) {
        this.startByte = startByte;
        this.endByte = endByte;
        this.status = SegmentStatus.PENDING;
    }

    public void start() {
        // HTTP Range request
    }

    public void pause() {
        // ...
    }

    public void resume() {
        // ...
    }
}
package com.example.downloadmanager.segment;

import com.example.downloadmanager.model.DownloadSegment;

import java.util.Iterator;
import java.util.List;
```

```
public class SegmentIterator implements Iterator<DownloadSegment> {

    private List<DownloadSegment> segments;
    private int index = 0;

    public SegmentIterator(List<DownloadSegment> segments) {
        this.segments = segments;
    }

    @Override
    public boolean hasNext() {
        return index < segments.size();
    }

    @Override
    public DownloadSegment next() {
        return segments.get(index++);
    }
}
package com.example.downloadmanager.segment;

import com.example.downloadmanager.model.DownloadTask;

public class SegmentManager {

    private DownloadTask task;

    public SegmentManager(DownloadTask task) {
```

```
        this.task = task;
    }

    public SegmentIterator iterator() {
        return new SegmentIterator(task.getSegments());
    }

    public void startAll() {
        // пройтись по сегментах через iterator
    }
}
package com.example.downloadmanager.command;

public interface DownloadCommand {
    void execute();
}
package com.example.downloadmanager.command;

import com.example.downloadmanager.core.DownloadManager;

public class StartDownloadCommand implements DownloadCommand {

    private DownloadManager manager;
    private Long taskId;

    public StartDownloadCommand(DownloadManager manager, Long taskId) {
        this.manager = manager;
        this.taskId = taskId;
    }
```

```
    @Override
    public void execute() {
        manager.startDownload(taskId);
    }
}
package com.example.downloadmanager.observer;

public interface DownloadObserver {

    void onProgress(long downloaded, long total);

    void onCompleted();
}
package com.example.downloadmanager.observer;

public class ConsoleDownloadObserver implements DownloadObserver {

    @Override
    public void onProgress(long downloaded, long total) {
        // вивід у консоль
    }

    @Override
    public void onCompleted() {
        // повідомлення про завершення
    }
}
package com.example.downloadmanager.storage;
```

```
import com.example.downloadmanager.model.DownloadTask;

public interface LocalStorage {

    void saveTask(DownloadTask task);

    DownloadTask loadTask(Long id);
}
package com.example.downloadmanager.storage;

import com.example.downloadmanager.model.DownloadTask;

public class SQLiteStorage implements LocalStorage {

    public SQLiteStorage() {
        // ініціалізація з'єднання з SQLite
    }

    @Override
    public void saveTask(DownloadTask task) {
        // INSERT / UPDATE
    }

    @Override
    public DownloadTask loadTask(Long id) {
        // SELECT
        return null;
    }
```

```
}
package com.example.downloadmanager;

import com.example.downloadmanager.core.DownloadManager;
import com.example.downloadmanager.observer.ConsoleDownloadObserver;
import com.example.downloadmanager.storage.SQLiteStorage;

public class Main {

    public static void main(String[] args) {

        DownloadManager manager =
                new DownloadManager(new SQLiteStorage());

        manager.addObserver(new ConsoleDownloadObserver());

        // демонстрація архітектури
    }
}
```

Питання до лабораторної роботи

1. Що таке UML?

UML (Unified Modeling Language) — це уніфікована мова візуального моделювання, яка використовується для специфікації, візуалізації, проєктування та документування програмних систем, бізнес-процесів і технічних рішень.

2. Що таке діаграма класів UML?

Діаграма класів UML — це статична діаграма, яка відображає структуру системи у вигляді класів, їх атрибутів, методів та зв’язків між ними.

3. Які діаграми UML називають канонічними?

Канонічними діаграмами UML називають стандартні типи діаграм, визначені специфікацією UML, такі як діаграми класів, варіантів використання, послідовностей, станів, діяльності, компонентів та розгортання.

4. Що таке діаграма варіантів використання?

Діаграма варіантів використання (Use Case Diagram) — це UML-діаграма, яка відображає функціональні вимоги до системи та взаємодію користувачів (акторів) із системою.

5. Що таке варіант використання?

Варіант використання — це опис окремої функції або сервісу системи, який демонструє, як система реагує на дії користувача для досягнення певної мети.

6. Які відношення можуть бути відображені на діаграмі використання?

На діаграмі варіантів використання можуть бути відображені такі відношення:

асоціація (actor – use case);

включення (<<include>>);

розширення (<<extend>>);

узагальнення (generalization).

7. Що таке сценарій?

Сценарій використання — це текстовий опис послідовності дій користувача та системи, що деталізує виконання конкретного варіанта використання від початку до завершення.

8. Що таке діаграма класів?

Діаграма класів — це UML-діаграма, яка описує статичну структуру програмної системи, показуючи класи, їхні властивості, методи та взаємозв’язки між ними.

9. Які зв’язки між класами ви знаєте?

Основні типи зв’язків між класами:

асоціація

агрегація

композиція

наслідування (generalization)

реалізація (realization)

залежність (dependency)

10. Чим відрізняється композиція від агрегації?

Композиція — це сильний зв’язок «частина-ціле», при якому життєвий цикл частини залежить від цілого.

Агрегація — слабший зв’язок, де об’єкти можуть існувати незалежно один від одного.

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

На діаграмах класів:

агрегація позначається порожнім ромбом і означає незалежне існування об’єктів;

композиція позначається зафарбованим ромбом і означає спільний життєвий цикл об’єктів.

12. Що являють собою нормальні форми баз даних?

Нормальні форми баз даних — це правила організації таблиць, які мінімізують надлишковість даних та забезпечують їхню цілісність. Основними є перша, друга та третя нормальні форми (1НФ, 2НФ, 3НФ).

13. Що таке фізична модель бази даних? Логічна?

Логічна модель БД описує структуру даних на рівні таблиць, атрибутів та зв’язків без урахування конкретної СУБД.

Фізична модель БД визначає спосіб реалізації логічної моделі у конкретній СУБД, включаючи типи даних, індекси та обмеження.

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

Таблиці бази даних відповідають програмним класам предметної області, а їхні поля — атрибутам класів. Такий взаємозв’язок реалізується за допомогою шаблону Repository або ORM-технологій, що забезпечує узгодженість між моделлю даних і програмною логікою.

Висновки

У ході виконання даної лабораторної роботи було виконано повний цикл аналізу та проєктування програмної системи відповідно до заданої теми — менеджер завантажень файлів (Download Manager). Основну увагу було зосереджено на застосуванні мови UML для формалізації вимог, опису архітектури системи та проєктування її основних компонентів.

На початковому етапі було проаналізовано предметну область та сформовано вимоги до системи, що дозволило побудувати діаграму варіантів використання, яка відображає взаємодію користувача з менеджером завантажень. На основі цієї діаграми було визначено ключові функціональні можливості системи, зокрема додавання, запуск, зупинку та відновлення завантажень, перегляд статистики та керування обмеженнями швидкості. Для обраних варіантів використання були розроблені детальні сценарії, що описують послідовність дій користувача та системи, можливі виняткові ситуації та результати виконання.

Наступним кроком стало проєктування діаграми класів предметної області, яка дозволила структуровано відобразити основні сутності системи, їхні атрибути та зв’язки між ними. Під час проєктування було застосовано низку шаблонів проєктування, зокрема Command, Observer, Template Method та Repository, що забезпечує гнучкість, розширюваність та модульність архітектури системи. Окремо була побудована діаграма класів реалізованої частини системи, згенерована на основі вихідного коду мовою Java, що підтверджує відповідність проєктної моделі реальній програмній реалізації.

Важливою частиною лабораторної роботи стало проєктування бази даних менеджера завантажень. Було розроблено логічну модель БД, яка включає таблиці для зберігання завдань завантаження, статистики, сегментів файлів, подій керування та налаштувань обмеження швидкості. Така структура бази даних забезпечує цілісність інформації, підтримує відновлення завантажень та дозволяє ефективно аналізувати роботу системи. Взаємодія між програмними класами та таблицями БД реалізується за

допомогою шаблону Repository, що спрощує доступ до даних та відокремлює бізнес-логіку від логіки збереження.

У результаті виконання лабораторної роботи було закріплено практичні навички використання UML-діаграм для аналізу та проєктування програмних систем, а також поглиблено розуміння принципів об’єктно-орієнтованого проєктування та застосування шаблонів проєктування. Отримані результати можуть бути використані як основа для подальшої реалізації повнофункціонального менеджера завантажень або для розширення системи новими можливостями.

Таким чином, поставлену мету лабораторної роботи було досягнуто в повному обсязі, а отримані знання та навички є важливими для подальшого вивчення дисципліни та практичного застосування в процесі розробки складних програмних систем.