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 № 3

Технології розробки програмного забезпечення

“ Основи проектування розгортання”

“Download manager”

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

Київ 2025

Тема: Основи проектування розгортання

Мета: Навчитися проєктувати діаграми розгортання та компонентів для системи що проєктується, а також розробляти діаграми взаємодії, а саме діаграми послідовностей, на основі сценаріїв зроблених в попередній лабораторній роботі.

Зміст

Завдання 2
Теоретичні відомості 3
Тема 4
Хід роботи 5
Діаграма розгортання 5
Опис діаграми розгортання 5
Опис вузлів діаграми розгортання 6
Діаграма компонентів 7
Діаграми послідовностей 9
Висновок: 12
Лістинг коду 12
Відповіді на контрольні питання 25

Завдання

• Ознайомитись з короткими теоретичними відомостями.

• Проаналізувати діаграми створені в попередній лабораторній роботі а також тему системи та спроєктувати діаграму розгортання використання відповідно до обраної теми лабораторного циклу.

• Розробити діаграму компонентів для проєктованої системи.

• Розробити діаграму розгортання для проєктованої системи.

• Розробити як мінімум дві діаграми послідовностей для сценаріїв прописаних в попередній лабораторній роботі.

• На основі спроєктованих діаграм розгортання та компонентів доопрацювати програмну частину системи. Реалізація системи, додатково до попередньої реалізації, повинна містити як мінімум дві візуальні форми. В системі вже повинен бути повністю реалізована архітектура (повний цикл роботи з даними від вводу на формі до збереження їх в БД і подальшій виборці з БД та відображенням на UI).

• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму розгортання з описом, діаграму компонентів системи з описом, діаграми послідовностей, а також вихідний код системи, який було додано в цій лабораторній роботі.

Теоретичні відомості

Діаграми компонентів — показують модулі (компоненти), їхні залежності і артефакти (.jar, таблиці, html). Використовуються для логічного/фізичного поділу системи.

Діаграма компонентів UML є представленням проєктованої системи, розбитої на окремі модулі. Залежно від способу поділу на модулі розрізняють три види діаграм компонентів:

• логічні;

• фізичні;

• виконувані.

Коли використовують логічне розбиття на компоненти, то у такому разі проєктовану систему віртуально уявляють як набір самостійних, автономних модулів (компонентів), що взаємодіють між собою.

Діаграми розгортання — показують фізичні вузли (devices) і середовища виконання (execution environments), на яких розгортаються артефакти; зв’язки зазначають протоколи (HTTP, SQL/ODBC).

Діаграми розгортання представляють фізичне розташування системи, показуючи, на якому фізичному обладнанні запускається та чи інша складова програмного забезпечення.

Головними елементами діаграми є вузли, пов'язані інформаційними шляхами. Вузол (node) – це те, що може містити програмне забезпечення. Вузли бувають двох типів. Пристрій (device) – це фізичне обладнання: комп'ютер або пристрій, пов'язаний із системою. Середовище виконання (execution environment) – це програмне забезпечення, яке саме може включати інше програмне забезпечення, наприклад операційну систему або процес-контейнер (наприклад, вебсервер).

Діаграми послідовностей — моделюють часову послідовність повідомлень

між акторами/об’єктами (HTTP POST/GET: Browser – Server DB – Browser).

Діаграма послідовностей (Sequence Diagram) – це один із типів діаграм у моделюванні UML (Unified Modeling Language), який використовується для моделювання взаємодії між об'єктами системи у певній послідовності часу. Вона відображає, як об'єкти обмінюються повідомленнями, показуючи порядок і логіку виконання операцій. Діаграма складається з таких основних елементів:

Актори (Actors): Зазвичай позначаються піктограмами або назвами. Це користувачі чи інші системи, які взаємодіють із системою. Актори можуть бути

Об'єкти або класи: Розміщуються горизонтально на діаграмі. Вони позначаються прямокутниками з іменем об'єкта або класу під прямокутником. Кожен об'єкт має «життєвий цикл», який представлений вертикальною пунктирною лінією (лінія життя).

Повідомлення: Це лінії зі стрілками, які з'єднують об'єкти. Вони показують передачу повідомлень чи виклик методів. Стрілка може бути синхронною (звичайна стрілка) або асинхронною (лінія з відкритим трикутником) та з пунктирною лінією, що показує повернення результату.

Активності: Вказують періоди, протягом яких об'єкт виконує певну дію. На діаграмі це позначається прямокутником, накладеним на лінію життя. Контрольні структури: Використовуються для відображення умов, циклів або альтернативних сценаріїв. Наприклад, блоки "alt" (альтернатива) або "loop" (цикл).

Тема

26. Download Manager

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

Менеджер завантажень призначений для скачування файлів з мережі Інтернет за протоколами **HTTP/HTTPS** з підтримкою запуску, зупинки та

відновлення завантажень. Система забезпечує багатопотокове завантаження файлів і керування швидкістю передачі даних між активними завантаженнями.

Менеджер зберігає статистику процесу завантаження, зокрема швидкість, обсяг переданих даних і час виконання, та надає можливість перегляду історії завантажень. Архітектура системи побудована з використанням шаблонів проєктування, що забезпечує гнучкість, розширюваність і можливість інтеграції з веб-браузерами.

Хід роботи

Діаграма розгортання

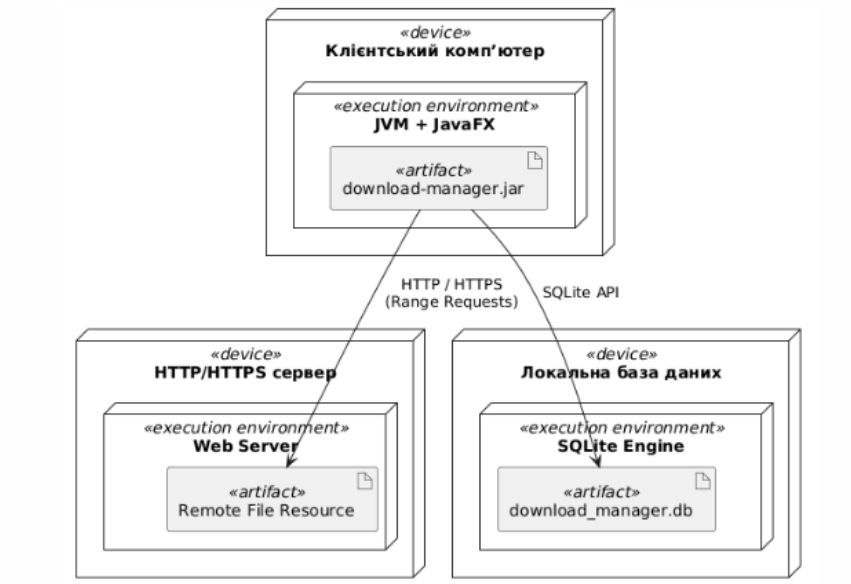

Рис. 1 – Діаграма розгортання

Опис діаграми розгортання

Система Download Manager є десктопним застосунком, який працює на комп'ютері користувача та взаємодіє з віддаленими HTTP/HTTPS серверами для завантаження файлів. Усі дані про завантаження зберігаються локально у вбудованій базі даних SQLite.

Діаграма розгортання складається з трьох основних вузлів:

1. Клієнтський комп'ютер користувача
2. Віддалений HTTP/HTTPS сервер
3. Локальне сховище даних (SQLite)

Опис вузлів діаграми розгортання

Вузол 1. «Клієнтський комп’ютер»

(Клієнтська сторона)

Пристрій (<<device>>)

Client PC — персональний комп’ютер користувача під управлінням операційної системи
(Windows / Linux / macOS).

Середовище виконання (<<execution environment>>)

- Java Virtual Machine (JVM) — середовище виконання Java-застосунку.
- JavaFX Runtime — середовище для запуску графічного користувацького інтерфейсу.

Артефакт (<<artifact>>)

- download-manager.jar — десктопний застосунок Download Manager, який:
  - надає користувацький інтерфейс (JavaFX),
  - керує завантаженнями файлів,
  - реалізує сегментне (P2P-подібне) HTTP-завантаження,
  - взаємодіє з локальною базою даних SQLite.

Вузол 2. «HTTP/HTTPS сервер»

(Зовнішнє джерело даних)

Пристрій (<<device>>)

Remote Web Server — віддалений сервер в мережі Інтернет.

Середовище виконання (<<execution environment>>)

- Web Server / HTTP Service — програмне забезпечення сервера, яке підтримує протоколи HTTP/HTTPS та механізм HTTP Range Requests.

Артефакт (<<artifact>>)

- Remote File Resource — файл або набір файлів, доступних для завантаження по HTTP/HTTPS.

Даний вузол не є частиною системи Download Manager, але використовується як джерело даних.

Вузол 3. «Локальна база даних»

(Рівень даних)

Пристрій (<<device>>)

Local Storage — файлове сховище на комп’ютері користувача.

Середовище виконання (<<execution environment>>)

- SQLite Engine — вбудована реляційна СУБД, що працює локально без серверної частини.

Артефакт (<<artifact>>)

- download_manager.db — файл бази даних SQLite, який містить:
  - інформацію про завантаження,
  - сегменти файлів,
  - статистику та події,
  - стан завантажень для відновлення (resume).

Комунікація між вузлами

- Клієнтський комп’ютер → HTTP/HTTPS сервер
  - Протокол: HTTP / HTTPS
  - Тип взаємодії: HTTP Range Requests
  - Призначення: сегментне завантаження файлів (P2P-подібний підхід).
- Клієнтський комп’ютер → Локальна база даних
  - Протокол: SQLite API / JDBC
  - Призначення: збереження та відновлення стану завантажень.

Діаграма компонентів

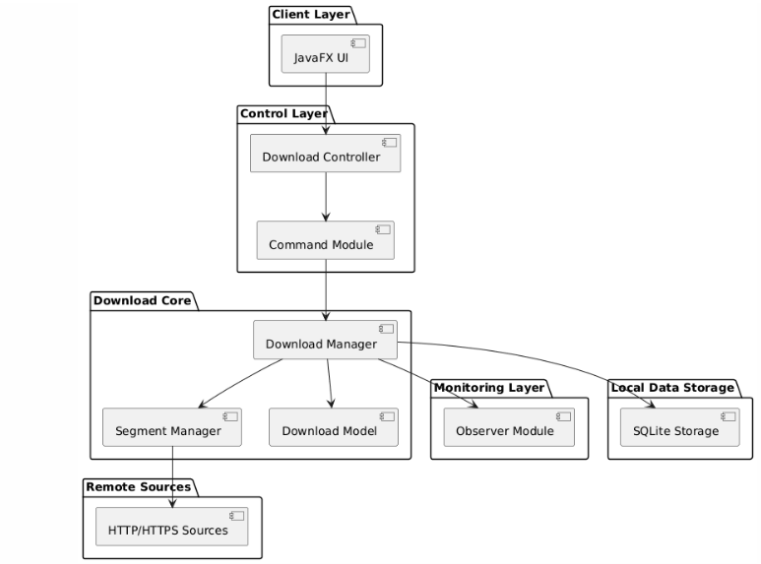

Рис.2 – Діаграма компонентів

Рівень Клієнта (Client)

Даний рівень представляє комп’ютер користувача, на якому виконується

десктопний застосунок Download Manager. Саме на цьому рівні відбувається взаємодія з користувачем, ініціація завантажень та відображення їх поточного стану.

User Interface (JavaFX UI)

Компонент користувацького інтерфейсу, реалізований за допомогою JavaFX. Він надає користувачеві можливість:

- додавати URL файлів для завантаження;
- запускати, призупиняти та відновлювати завантаження;
- переглядати прогрес і статистику.

UI не містить бізнес-логіки та передає всі дії користувача до контролюючого компонента.

Рівень Керування (Control Layer)

Даний рівень відповідає за обробку дій користувача та перетворення їх у керуючі команди для системи завантаження.
Download Controller

Компонент-посередник між користувацьким інтерфейсом та ядром системи. Він приймає події з UI та створює відповідні об’єкти команд.
Command Module

Модуль, що реалізує шаблон проєктування Command. Кожна дія користувача (запуск, пауза, відновлення, зупинка завантаження) інкапсулюється в окремий об’єкт команди, який виконується ядром системи.

Рівень Ядра Завантаження (Download Core)

Цей рівень реалізує основну бізнес-логіку менеджера завантажень та координує роботу всіх внутрішніх компонентів.
Download Manager
Центральний компонент системи. Він керує життєвим циклом завдань завантаження, обробляє команди, ініціює сегментне завантаження та повідомляє інші компоненти про зміни стану.

Segment Manager

Компонент, відповідальний за розбиття файлу на сегменти та керування їх завантаженням. Він організовує паралельні HTTP range-з’єднання та забезпечує перебір сегментів за допомогою шаблону Iterator.

Download Model

Набір доменних сутностей (DownloadTask, DownloadSegment), що представляють структуру завантаження. Реалізує шаблон Composite, де завдання складається з множини сегментів.

Рівень Моніторингу та Повідомлень (Monitoring & Observer)
Цей рівень відповідає за відстеження стану завантажень та автоматичне інформування зацікавлених компонентів.
Observer Module

Модуль, що реалізує шаблон Observer. Він дозволяє незалежним спостерігачам (UI, модуль статистики) отримувати повідомлення про прогрес та завершення завантажень без жорсткого зв’язування з ядром системи.

Рівень Віддалених Джерел (Remote Sources)

Цей рівень представляє зовнішні ресурси, які не входять до складу системи, але є джерелами даних для завантаження.
HTTP/HTTPS Sources

Віддалені веб-сервери або CDN-вузли, що підтримують HTTP/HTTPS та range-запити. Кожне джерело розглядається як рівноправний учасник процесу сегментного завантаження, що відповідає P2P-підходу на рівні мережевих з’єднань.

Рівень Локального Зберігання Даних (Local Data Storage)
Цей рівень забезпечує збереження стану завантажень та статистики для можливості відновлення процесу після перезапуску застосунку.

SQLite Storage
Компонент доступу до локальної вбудованої бази даних SQLite. Він відповідає за збереження:

- завдань завантаження;
- сегментів файлів;
- статистики та подій.

Діаграми послідовностей

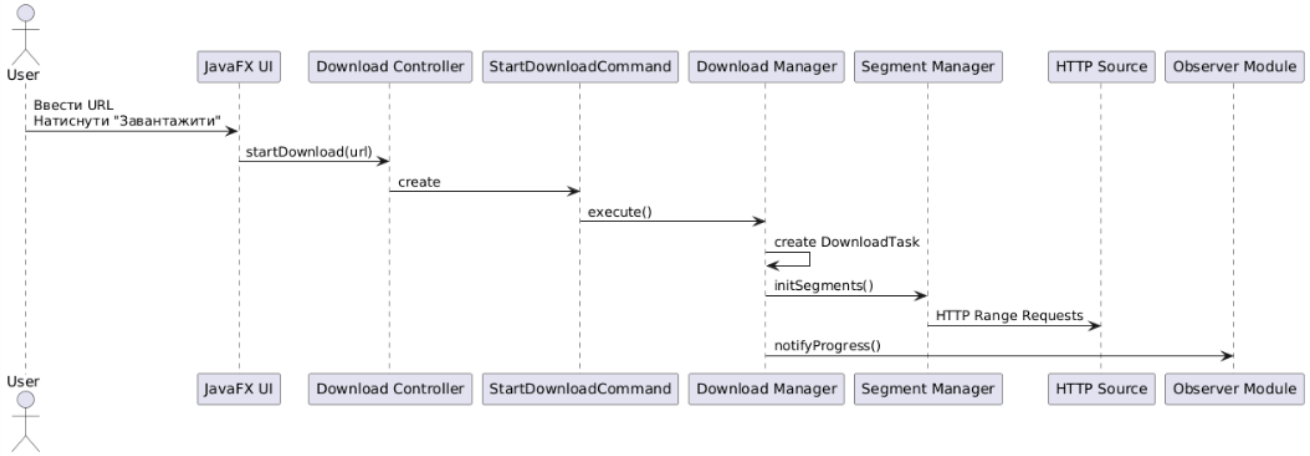

Рис.3 – Діаграма послідовностей для запуску нового завантаження

Сценарій: Додавання та запуск нового завантаження
Призначення
Діаграма відображає процес додавання нового завдання завантаження та запуск сегментного HTTP-завантаження.

Основний перебіг подій
- Користувач у графічному інтерфейсі вводить URL файлу.
- User Interface передає дію до Download Controller.
- Download Controller створює команду StartDownloadCommand.
- Команда передає запит до Download Manager.
- Download Manager створює об’єкт DownloadTask.
- Download Manager ініціалізує Segment Manager.
- Segment Manager розбиває файл на сегменти.
- Для кожного сегмента ініціюється HTTP range-завантаження.
- Download Manager сповіщає Observer про початок завантаження.

Альтернативний перебіг подій
- Якщо URL некоректний або недоступний:
- Download Manager не створює завдання.
- Observer отримує повідомлення про помилку.
- Процес завантаження завершується.

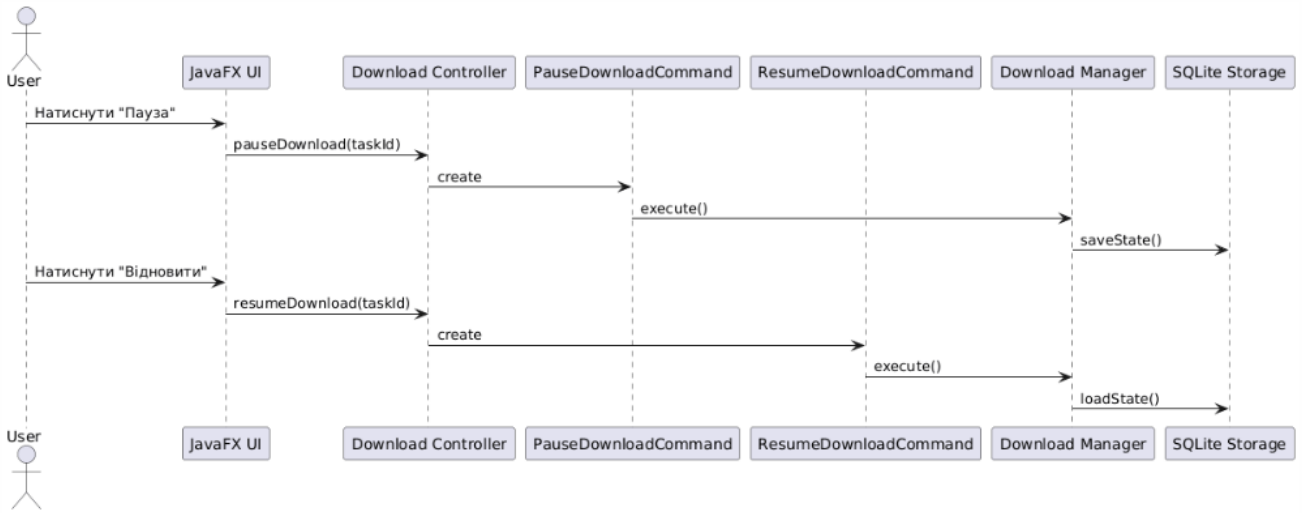

Рис. 4 – Діаграма для призупинення завантаження

Сценарій: Призупинення та відновлення завантаження
Призначення

Діаграма демонструє процес призупинення активного завантаження та подальшого відновлення з використанням збереженого стану.

Основний перебіг подій
- Користувач у UI натискає кнопку «Пауза».
- UI передає подію до Download Controller.
- Download Controller створює PauseDownloadCommand.

- Команда викликає метод pauseDownload() у Download Manager.
- Download Manager зупиняє всі активні сегменти.
- Поточний стан зберігається у SQLite Storage.
- Користувач натискає кнопку «Відновити».
- Download Controller створює ResumeDownloadCommand.
- Download Manager зчитує збережений стан.
- Сегментне завантаження продовжується з попередньої позиції.

Альтернативний перебіг подій
- Якщо дані стану відсутні або пошкоджені:
- Download Manager не може відновити завантаження.
- Observer отримує повідомлення про помилку відновлення.

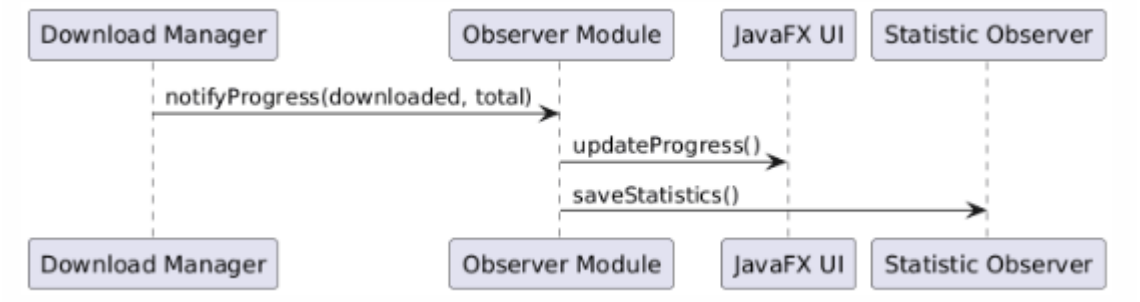

Рис. 5 – Діаграма для призупинення завантаження

Сценарій: Оновлення прогресу та статистики завантаження

Призначення
Діаграма ілюструє механізм автоматичного оновлення прогресу завантаження з використанням шаблону Observer.

Основний перебіг подій
- Під час завантаження сегмента змінюється кількість завантажених даних.
- Download Manager фіксує зміну прогресу.
- Download Manager викликає метод notifyProgress().
- Observer Module отримує оновлення.
- UI оновлює відображення прогресу.
- Statistic Observer зберігає статистичні дані.

Альтернативний перебіг подій
- Якщо Observer не підписаний:
- Download Manager не виконує сповіщення.
- Процес завантаження продовжується без оновлення UI.

Висновок:

У ході виконання лабораторної роботи було спроєктовано та частково реалізовано архітектуру десктопного менеджера завантажень файлів, що працює за протоколами HTTP/HTTPS та підтримує сегментне завантаження з можливістю відновлення процесу.

Було побудовано основні UML-діаграми системи, зокрема діаграми варіантів використання, класів, компонентів, послідовностей та розгортання, що дозволило формалізувати структуру та взаємодію компонентів.

У коді системи закладено каркас для реалізації шаблонів проєктування Command, Iterator, Observer, Composite та Template Method, а також передбачено використання локальної вбудованої бази даних SQLite для збереження стану завантажень. Реалізація виконана на рівні архітектурної заготовки, що відповідає вимогам поточного етапу лабораторної роботи та створює основу для подальшого функціонального розширення системи.

Лістинг коду

```
// DownloadStatus.java
public enum DownloadStatus {
    NEW,
    DOWNLOADING,
    PAUSED,
    COMPLETED,
    ERROR
}

// SegmentStatus.java
public enum SegmentStatus {
    PENDING,
    DOWNLOADING,
    COMPLETED,
```

```
    ERROR
}

// DownloadSegment.java
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 download
        // ...
    }

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

    public void resume() {
```

```
        // ...
    }
}

// DownloadTask.java
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;
    }
}


// SegmentIterator.java
import java.util.List;

public class SegmentIterator {

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

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

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

```
    public DownloadSegment next() {
        return segments.get(index++);
    }
}

// SegmentManager.java
public class SegmentManager {

    private DownloadTask task;

    public SegmentManager(DownloadTask task) {
        this.task = task;
    }

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

    public void startAllSegments() {
        // iterate through segments
        // ...
    }
}

// DownloadCommand.java
```

```
public interface DownloadCommand {
    void execute();
}

// StartDownloadCommand.java
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);
    }
}

// PauseDownloadCommand.java
public class PauseDownloadCommand implements DownloadCommand {

    private DownloadManager manager;
    private Long taskId;
```

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

    @Override
    public void execute() {
        manager.pauseDownload(taskId);
    }
}

// ResumeDownloadCommand.java
public class ResumeDownloadCommand implements DownloadCommand {

    private DownloadManager manager;
    private Long taskId;

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

    @Override
    public void execute() {
        manager.resumeDownload(taskId);
```

```
    }
}

// StopDownloadCommand.java
public class StopDownloadCommand implements DownloadCommand {

    private DownloadManager manager;
    private Long taskId;

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

    @Override
    public void execute() {
        manager.stopDownload(taskId);
    }
}


// DownloadObserver.java
public interface DownloadObserver {

    void onProgress(long downloaded, long total);
```

```
    void onCompleted();
}

// ConsoleDownloadObserver.java
public class ConsoleDownloadObserver implements DownloadObserver {

    @Override
    public void onProgress(long downloaded, long total) {
        // print progress
    }

    @Override
    public void onCompleted() {
        // download finished
    }
}


// DownloadManager.java
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) {
    // initialize task
    // start segments
}

public void pauseDownload(Long taskId) {
    // pause segments
}

public void resumeDownload(Long taskId) {
    // resume segments
}

public void stopDownload(Long taskId) {
    // stop download
}

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();
        }
    }
}


// LocalStorage.java
public interface LocalStorage {

    void saveTask(DownloadTask task);

    DownloadTask loadTask(Long id);
}

// SQLiteStorage.java
public class SQLiteStorage implements LocalStorage {
```

```
    public SQLiteStorage() {
        // initialize SQLite connection
    }

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

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


// HttpRangePeer.java
public class HttpRangePeer {

    public void downloadSegment(String url, long start, long end) {
        // HTTP Range request
        // ...
```

```
    }
}


// UserInterface.java
public class UserInterface {

    private DownloadCommand startCommand;
    private DownloadCommand pauseCommand;
    private DownloadCommand resumeCommand;
    private DownloadCommand stopCommand;

    public void clickStart() {
        startCommand.execute();
    }

    public void clickPause() {
        pauseCommand.execute();
    }

    public void clickResume() {
        resumeCommand.execute();
    }

    public void clickStop() {
        stopCommand.execute();
```

```
    }
}
```

```
// Main.java
public class Main {

    public static void main(String[] args) {

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

        manager.addObserver(new ConsoleDownloadObserver());

        // application start
        // ...
    }
}
```

Відповіді на контрольні питання

1. Що собою становить діаграма розгортання?

Діаграма розгортання (Deployment Diagram) — це UML-діаграма, яка відображає фізичну архітектуру системи, тобто розміщення програмних компонентів на апаратних вузлах та їхню взаємодію через мережеві з’єднання. Вона показує, де і як саме виконуються частини програмної системи.

2. Які бувають види вузлів на діаграмі розгортання?

На діаграмі розгортання використовуються такі види вузлів:

Пристрій (<<device>>) — фізичний обчислювальний пристрій (комп’ютер, сервер, мобільний пристрій).

Середовище виконання (<<execution environment>>) — програмне середовище, у якому виконуються компоненти (операційна система, JVM, контейнер застосунків).

Вузол — узагальнене поняття, що може представляти як фізичні, так і логічні елементи інфраструктури.

3. Які бувають зв’язки на діаграмі розгортання?

На діаграмі розгортання використовуються:

Комунікаційні зв’язки — відображають мережеву або логічну взаємодію між вузлами (HTTP, HTTPS, TCP/IP, SQL тощо).

Зв’язки вкладеності — показують, що один вузол розміщений всередині іншого (наприклад, середовище виконання всередині пристрою).

4. Які елементи присутні на діаграмі компонентів?

Основними елементами діаграми компонентів є:

Компоненти — незалежні модулі системи з чітко визначеною відповідальністю.

Інтерфейси — точки взаємодії компонентів (надані та необхідні).

Порти — місця підключення інтерфейсів.

Пакети — логічне групування компонентів.

5. Що становлять собою зв’язки на діаграмі компонентів?

Зв’язки на діаграмі компонентів показують:

Залежності (dependency) — один компонент використовує функціональність іншого.

З’єднання через інтерфейси — взаємодію компонентів через чітко визначені контракти.

Реалізацію інтерфейсів — компонент реалізує певний інтерфейс.

6. Які бувають види діаграм взаємодії?

До діаграм взаємодії в UML належать:

Діаграма послідовностей (Sequence Diagram)

Діаграма комунікацій (Communication Diagram)

Діаграма часових характеристик (Timing Diagram)

Діаграма огляду взаємодії (Interaction Overview Diagram)

7. Для чого призначена діаграма послідовностей?

Діаграма послідовностей використовується для опису динамічної поведінки системи, тобто порядку обміну повідомленнями між об’єктами або компонентами у часі під час виконання певного сценарію.

8. Які ключові елементи можуть бути на діаграмі послідовностей?

Основними елементами діаграми послідовностей є:

Учасники (Lifelines) — об’єкти, актори або компоненти.

Повідомлення — виклики методів або передача даних.

Активації (Activation bars) — період активності об’єкта.

Фрагменти керування — alt, opt, loop для альтернативних і повторюваних сценаріїв.

9. Як діаграми послідовностей пов’язані з діаграмами варіантів використання?

Діаграми послідовностей деталізують варіанти використання, показуючи, як саме реалізується конкретний сценарій взаємодії між користувачем і системою, описаний на діаграмі варіантів використання.

10. Як діаграми послідовностей пов’язані з діаграмами класів?

Діаграми послідовностей використовують класи та методи, визначені на діаграмі класів, і показують їхню взаємодію в динаміці. Таким чином, діаграма класів описує статичну структуру системи, а діаграма послідовностей — її поведінку у часі.