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

з дисципліни "Технології розроблення програмного забезпечення"

Виконав

студент групи ІА–33:

Китченко Д.В.

Перевірив:

Мягкий М.Ю.

Київ 2025

Зміст

Вступ

У даній лабораторній роботі розглядаються основи проектування програмного забезпечення на прикладі розробки архітектури для застосунку “Архіватор”. Метою роботи є опанування ключових інструментів та методологій проектування, зокрема використання мови UML для візуалізації архітектури, розробки сценаріїв використання для визначення функціональних вимог, а також застосування патернів проектування для побудови гнучкої та розширюваної системи.

У ході роботи було спроектовано систему з підтримкою різних форматів архівів, реалізовано механізм взаємодії з базою даних через патерн Repository та побудовано логіку взаємодії компонентів за допомогою патернів Strategy, Factory Method та Facade. Результатом роботи є детальний проект архітектури, готовий для подальшої реалізації.

Архіватор (strategy, adapter, factory method, facade, visitor, p2p) Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, .rar, .ace) – додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.`

Теоретичні відомості

Процес розробки програмного забезпечення є складною діяльністю, що вимагає системного підходу на всіх етапах життєвого циклу. Ключовим етапом є об'єктно-орієнтований аналіз та проектування (ООАП), який дозволяє створювати надійні, гнучкі та підтримувані системи. У даній лабораторній роботі для проектування системи “Архіватор” були застосовані фундаментальні концепції ООАП, зокрема мова UML та принципи проектування баз даних.

Уніфікована мова моделювання (UML) є загальновизнаним стандартом для візуального моделювання програмних систем. Вона надає набір графічних нотацій для специфікації, візуалізації та документування архітектури ПЗ. Процес проектування за допомогою UML можна розглядати як ієрархічний перехід від більш загальних, концептуальних моделей до деталізованих логічних та фізичних уявлень.

- Концептуальний рівень представляє систему з точки зору користувача та бізнес-логіки, не вдаючись у деталі реалізації. На цьому рівні будується діаграма варіантів використання.
- Логічний рівень описує статичну структуру та динамічну поведінку системи. До цього рівня відносяться діаграми класів, послідовностей та станів.
- Фізичний рівень відображає, як компоненти системи розгортаються на апаратному забезпеченні, і описується діаграмами компонентів та розгортання.

Діаграма варіантів використання (Use Case Diagram) є відправною точкою в аналізі вимог. Вона фіксує функціональність системи та її межі. Основними елементами діаграми є:

Актор (Actor): Будь-яка зовнішня сутність, що взаємодіє із системою (користувач, інша система). В проекті "Архіватор" єдиним актором є "Користувач".

Варіант використання (Use Case): Опис послідовності дій, які система виконує у відповідь на запит актора для досягнення певної мети. Приклади: "Створити архів", "Додати файли".

Відносини: Зв'язки, що описують взаємодію. В роботі використано асоціацію (показує, що актор ініціює варіант використання) та включення (<<include>>) (показує, що один варіант використання обов'язково включає в себе поведінку іншого, наприклад, "Додати файли" включає "Відкрити архів"). <<extend>> - додатковий крок, який може виконуватись за потреби.

Діаграма варіантів використання надає лише загальне уявлення про функціонал. Для усунення неоднозначності та надання чітких специфікацій для розробників кожен варіант використання деталізується за допомогою сценарію використання. Це текстовий, покроковий опис взаємодії "актор-система", що включає:

Передумови: Умови, що мають бути виконані до початку сценарію.

Постумови: Стан системи після успішного або неуспішного завершення.

Основний потік подій: Послідовність кроків при ідеальному виконанні.

Винятки: Опис того, як система реагує на помилки або нестандартні ситуації.

Діаграма класів є основним інструментом для моделювання статичної структури системи. Вона візуалізує класи, їхні атрибути (дані) та операції (методи), а також зв'язки між ними.

Атрибути та операції мають рівні видимості (+ public, - private, # protected), що відображає принципи інкапсуляції.

Відносини між класами є ключовим елементом діаграми:

Асоціація: Загальний зв'язок, що показує, що об'єкти класів взаємодіють.

Узагальнення (успадкування): Відношення "є різновидом" (is-a), коли дочірній клас успадковує структуру та поведінку батьківського.

Агрегація та композиція: Відношення "частина-ціле" (has-a). Композиція (зафарбований ромб) позначає жорсткий зв'язок, де "частина" не може існувати без "цілого" (наприклад, ArchivedFolder та його ArchiveItem).

Для зберігання даних системи була спроектована логічна модель бази даних. На відміну від фізичної моделі, логічна описує дані у вигляді таблиць, полів, ключів та зв'язків. Проєктування роблять через нормалізацію - щоб уникнути дублювання та помилок. У даній роботі таблиці спроектовані з

урахуванням перших трьох нормальних форм (1НФ, 2НФ, 3НФ), що є стандартом для більшості реляційних баз даних. Структура БД та зв'язки між таблицями були візуалізовані за допомогою MS Access.

Хід роботи

1) Аналіз вимог та проектування діаграми варіантів використання

На першому етапі проектування було проведено аналіз функціональних вимог до системи. Було визначено, що система повинна надавати користувачу повний набір інструментів для роботи з архівами.

На основі цього аналізу було виділено одного ключового актора:

Користувач - особа, яка безпосередньо взаємодіє із застосунком для створення, перегляду та маніпуляції архівами.

Було визначено наступні варіанти використання, що описують основні функції системи:

Створити архів

Відкрити/переглянути архів

Розархівувати файли

Додати файли/папки

Видалити файли/папки

Редагувати метадані

Протестувати архів

Перевірити контрольні суми

Розбити архів на частини

Зібрати частини

Для відображення логічних залежностей між операціями було використано відношення включення (<<include>>). Наприклад, для виконання операцій додавання, видалення чи розпакування файлів, архів обов'язково має бути попередньо відкритий. Результатом цього етапу є діаграма варіантів використання, наведена на Рис. 1.

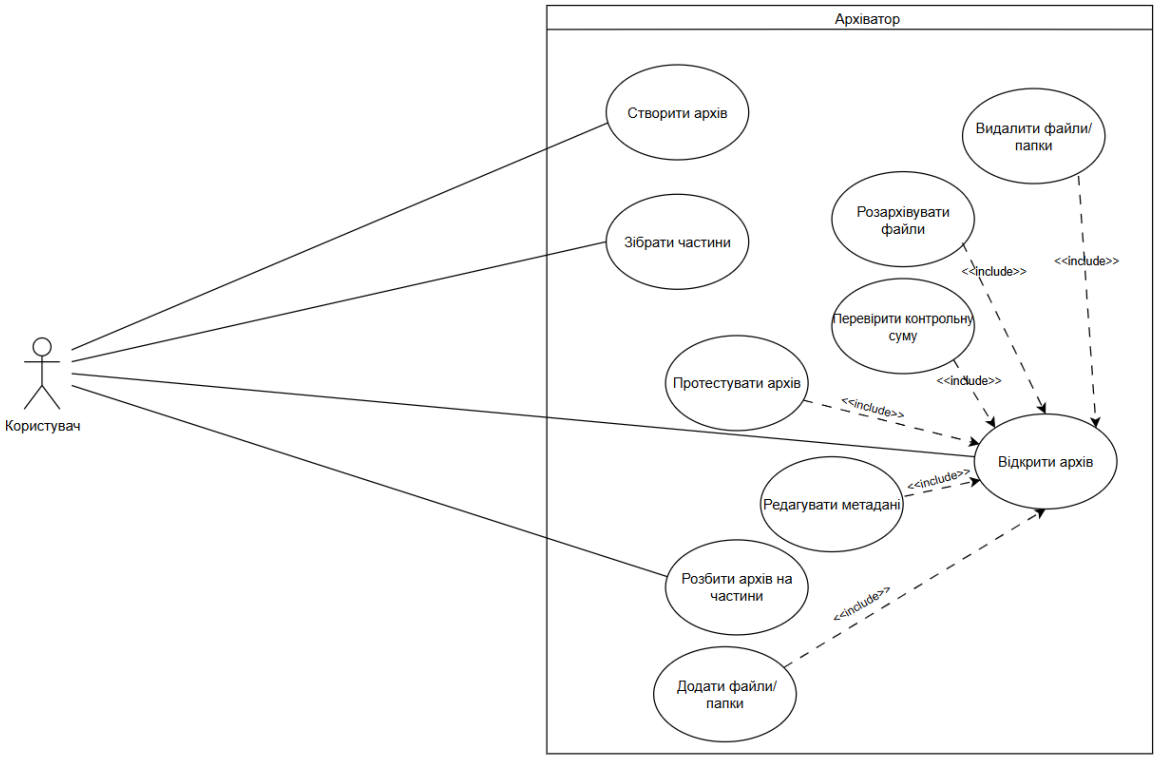

Рис. 1. Діаграма варіантів використання системи "Архіватор"

Я також логічно розділив всі операції на дві групи:

- Незалежні операції, такі як "Створити архів", "Зібрати частини" та "Розбити архів", які працюють з файлами на диску "ззовні". Окремо варто зазначити, що такі варіанти використання, як "Створити архів", "Розбити архів на частини" та "Зібрати частини", є незалежними (самостійними) операціями. Вони не включають (<<include>>) і не розширюють (<<extend>>) інші базові функції, оскільки представляють собою унікальні, повноцінні сценарії роботи з файлами на диску. Зокрема, "Зібрати частини" не є розширенням для "Відкрити архів". Це окрема ціль користувача, яка полягає у склеюванні файлів-частин в один цілий архів, а не в перегляді вмісту. Користувач свідомо обирає цю дію, тому вона винесена в окремий, незалежний варіант використання, який ініціюється напряму.
- Залежні операції, які працюють з вмістом вже відкритого архіву.

2) Розробка сценаріїв варіантів використання

Обрано три варіанти використання (Створити архів, Відкрити архів, Додати файли) і для кожного описано покроковий сценарій (передумови, основний потік, винятки, постумови).

Сценарій №1: Створення нового архіву

Передумови: Програму запущено.

Постумови:

Успіх: На диску створено новий архів заданого формату з вибраними файлами/папками.

Провал: Стан файлової системи не змінено

Взаємодіючі сторони: Користувач, Система “Архіватор”.

Короткий опис: Користувач формує архів, вибираючи вміст, ім’я, формат і місце збереження; система стискає й створює файл архіву.

Основний перебіг подій.

1. Користувач натискає “Створити”.
2. Система відкриває діалог вибору файлів/папок.
3. Користувач відмічає елементи та натискає “Далі”.
4. Система відкриває діалог збереження: ім’я архіву, формат (.zip/.tar.gz/.rar/.ace), місце, рівень стиснення.
5. Користувач підтверджує “Зберегти”.
6. Система валідує шлях/права/вільне місце, показує індикатор і стискає вміст.
7. Після завершення система показує “Архів успішно створено”.

Винятки

Виняток №1: На кроках 2/4 користувач натискає “Скасувати” -> операцію перервано, без змін.

Виняток №2: На кроці 6 Система виявляє, що немає місця. Процес переривається. Система виводить повідомлення про помилку "Недостатньо місця на диску”.

Виняток №3: На кроці 5 Користувач вводить ім'я файлу, який вже існує в цільовій теці. Система відображає стандартне діалогове вікно з питанням:

“Файл [ім'я_архіву] вже існує. Замінити його?”. Сценарій продовжується відповідно до вибору Користувача (Так/Ні).

Виняток №4: На кроці 5 Користувач вибирає тип архіву, для створення якого в Системі відсутні необхідні інструменти (наприклад, через відсутність ліцензії або бібліотеки). Система виводить повідомлення: “Створення архівів типу [тип] не підтримується цією версією програми”.

Виняток №5: На кроці 6 немає прав на запис у директорію -> повідомлення; запропонувати обрати інший шлях.

Сценарій №2: Розархівування файлів з архіву

Передумови: На диску існує файл архіву.

Постумови:

Успіх: Вибрані файли/теки з архіву з'явились у вказаній теці на диску.

Провал: Розпакування не завершено; деякі файли могли бути створені до моменту помилки – система показує їх перелік.

Взаємодіючі сторони: Користувач, система “Архіватор”.

Короткий опис: Користувач витягує один або декілька елементів із наявного архіву до вибраної папки.

Основний перебіг подій.

1. Користувач обирає “Розархівувати файли”.

2. Якщо архів не відкрито, система показує діалог вибору файлу архіву та завантажує його вміст.

3. Система показує вміст архіву у вигляді списку/дерева.

4. Користувач позначає потрібні елементи.

5.Система відкриває діалог вибору цільової теки з опціями(перезапис/пропуск конфліктів, зберегти структуру).

6. Користувач підтверджує “Розпакувати”.

7. Система перевіряє права доступу та вільне місце, виконує розпакування з індикатором прогресу.

8. Після завершення Система показує повідомлення “Розархівацію завершено успішно”.

Винятки

Виняток №1: На кроках 2/5 користувач натискає “Скасувати” -> операцію перервано, без змін.

Виняток №2: На кроці 7 виявлено брак вільного місця -> повідомлення, пропозиція вибрати іншу папку або зменшити обсяг.

Виняток №3: Немає прав на читання архіву/запис у ціль -> повідомлення; запропонувати інший шлях/папку.

Виняток №4: На кроці 5 (або 7, якщо політику не обрано) система виявляє, що у цільовій теці вже існує файл з таким самим ім’ям. Система показує діалог: “Файл “{ім’я}” вже існує. Замінити його?”

Варіанти: Так – перезаписати; Ні – пропустити.

Опція: Застосувати до всіх. Система продовжує згідно з вибором користувача.

Виняток №5: На кроці 2 або 7 система не може прочитати архів -> повідомлення: “Архів пошкоджений. Виконайте перевірку”.

Виняток №6: На кроці 7 користувач натискає “Скасувати” -> процес зупинено; частково створені файли залишаються, система показує список.

Виняток №7: На кроці 5 (або 7, якщо проблема виявлена під час запису) система визначає, що цільовий шлях для файла “{ім’я}” некоректний (заборонені символи, надто довга назва/шлях, обмеження файлової системи). Система показує діалог із варіантами: Вибрати іншу теку, Перейменувати, Пропустити цей елемент; опція “Застосувати до всіх подібних випадків”. Система виконує вибрану дію та продовжує операцію.

Сценарій №3: Перевірити контрольну суму (checksum)

Передумови: Користувач відкрив існуючий архів у програмі.

Постумови:

Успіх: Система показала звіт по вибраних елементах зі статусом OK/Невідповідність.

Провал: Перевірку не завершено через помилку. Стан архіву не змінився.

Короткий опис: Користувач запускає перевірку цілісності для вибраних файлів усередині архіву. Система обчислює їхні контрольні суми й порівнює з

еталонними (збереженими під час створення архіву); якщо еталону немає – показує обчислені значення.

Основний перебіг подій.

1. Користувач виділяє один або кілька файлів у вмісті архіву.
2. Натискає “Перевірити контрольну суму”.
3. Система для кожного вибраного елемента:

a) читає дані з архіву;

b) обчислює контрольну суму;

c) якщо в метаданих/супровідному файлі є еталон – порівнює з ним; якщо еталону нема – просто показує обчислену суму.

4. Система відображає підсумковий список елементів зі статусом OK або Невідповідність.

Винятки

Виняток №1: На кроці 2 або під час кроку 3 користувач натискає “Скасувати” -> операцію перервано, без змін.

Виняток №2: На кроці 3a помилка читання/пошкоджений архів -> елемент пропущено, інші тривають.

Виняток №3: На кроці 3c еталонної суми нема -> не помилка: показати лише обчислене значення.

Виняток №4: На кроці 3c еталон невірний/непідтримуваний -> запропонувати інший або продовжити без порівняння.

Виняток №5: На кроці 3a немає прав читання елемента -> повідомлення; запропонувати інший файл.

3) Проектування діаграми класів предметної області

На наступному етапі було створено модель предметної області, яка відображає основні сутності системи та зв’язки між ними, без урахування технічних деталей реалізації. Центральним елементом виступає клас Archive**,** що представляє архів як контейнер. Його вміст описується через ієрархію класів, успадкованих від абстрактного класу ArchiveItem. Клас ArchivedFolder

реалізує ієрархічну структуру (дерево), оскільки може містити як окремі файли (ArchivedFile), так і вкладені папки.

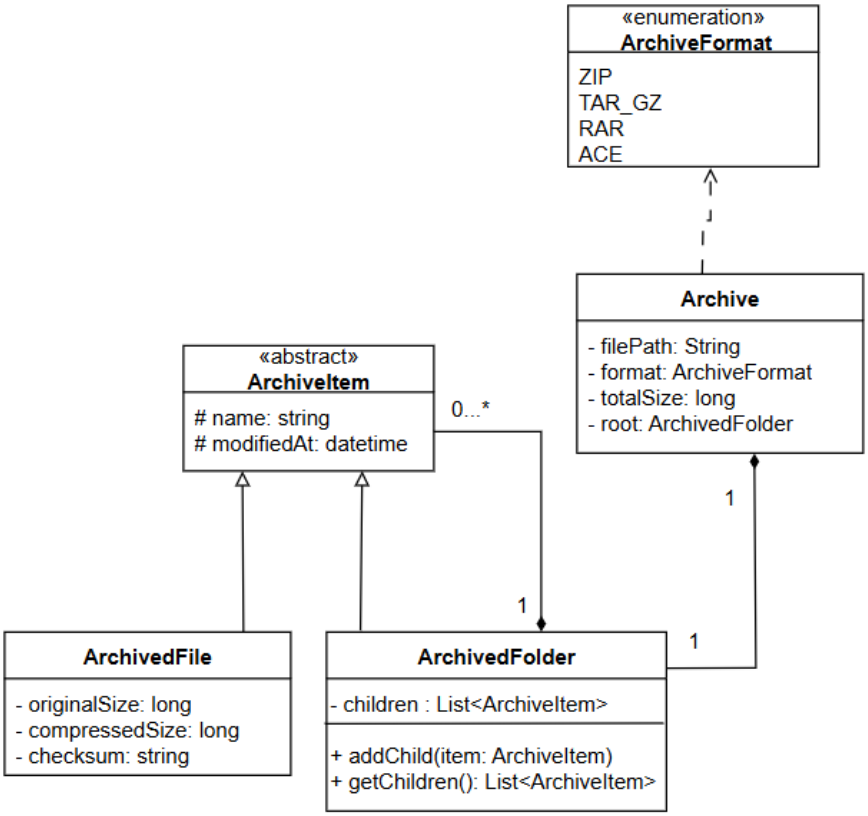

Рис. 2. Діаграма класів предметної області

4) Проектування структури бази даних та реалізація патерну Repository

Для зберігання метаданих про архіви та історії операцій було спроектовано реляційну базу даних. Вона складається з чотирьох основних таблиць:

ArchiveInfo: Зберігає загальну інформацію про кожен архів (шлях, формат, розмір, дати).

OperationLog: Виконує роль журналу, фіксуючи кожну операцію, виконану над архівом.

OperationDetail: Таблиця для деталізації операцій. Вона пов'язана з OperationLog відношенням "один-до-багатьох" і зберігає інформацію про кожен окремий файл, якого торкнулася операція. Це дозволяє вести детальний аудит.

ArchiveBookmark: Таблиця для реалізації функціоналу закладок. Вона дозволяє користувачу зберігати посилання на важливі файли чи папки всередині архівів для швидкого доступу.

Всі таблиці пов'язані логічними відношеннями, що забезпечують цілісність даних.

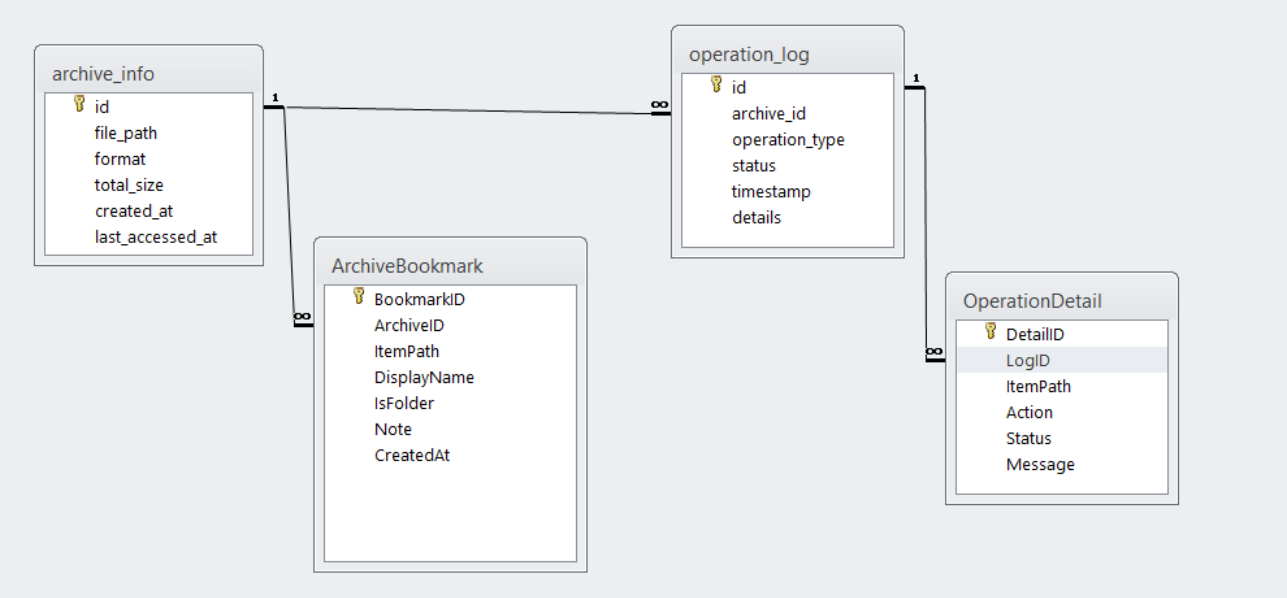

Рис. 3. Схема даних, реалізована в MS Access

Для взаємодії з цією базою даних було реалізовано патерн Repository. Він ізолює бізнес-логіку від деталей роботи з БД, надаючи інтерфейси IArchiveInfoRepository та IOperationLogRepository. У даній лабораторній роботі реалізації цих інтерфейсів (...Impl) імітують роботу з БД за допомогою колекцій в пам'яті.

5) Проектування діаграми класів реалізованої системи

Фінальна діаграма класів відображає повну архітектуру розробленої частини системи. Вона об'єднує класи предметної області, моделі даних, репозиторії та всі застосовані патерни проектування:

Facade: ArchiverFacade слугує єдиною точкою входу в систему.

Strategy: IArchiverStrategy та його реалізація ZipStrategy інкапсулюють логіку роботи з конкретним форматом архіву.

Factory Method: ArchiverStrategyFactory відповідає за створення необхідних об'єктів-стратегій.

Repository: I...Repository та ...RepositoryImpl забезпечують доступ до даних.

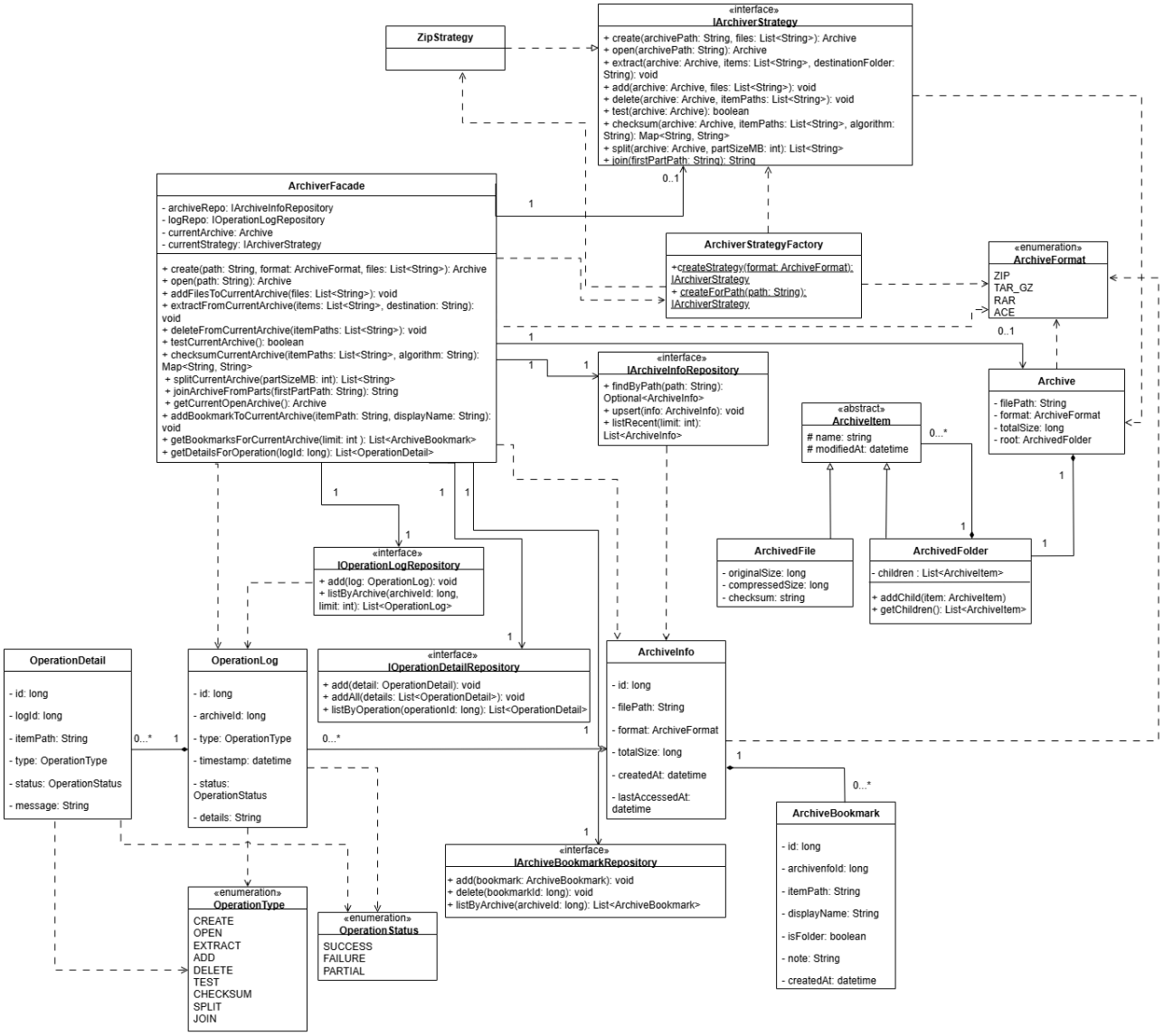

Рис. 4. Діаграма класів реалізованої системи

6) Лістинг коду реалізованих класів

Пакет model

Archive

```
package model;

import java.time.LocalDateTime;

public class Archive {
    private String filePath;
    private ArchiveFormat format;
    private long totalSize;
    private ArchivedFolder rootFolder;

    public Archive(String filePath, ArchiveFormat format) {
        this.filePath = filePath;
        this.format = format;
        this.rootFolder = new ArchivedFolder("/", LocalDateTime.now());
    }
```

```
    public String getFilePath() {
        return filePath;
    }

    public ArchiveFormat getFormat() {
        return format;
    }

    public long getTotalSize() {
        return totalSize;
    }

    public void setTotalSize(long totalSize) {
        this.totalSize = totalSize;
    }

    public ArchivedFolder getRootFolder() {
        return rootFolder;
    }
}
```

ArchiveInfo

```
package model;

import java.time.LocalDateTime;

public class ArchiveInfo {
    private long id;
    private String filePath;
    private String format;
    private long totalSize;
    private LocalDateTime createdAt;
    private LocalDateTime lastAccessedAt;
    public ArchiveInfo(long id, String filePath, String format,
                       long totalSize, LocalDateTime createdAt,
                       LocalDateTime lastAccessedAt) {
        this.id = id;
        this.filePath = filePath;
        this.format = format;
        this.totalSize = totalSize;
        this.createdAt = createdAt;
        this.lastAccessedAt = lastAccessedAt;

    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getFilePath() {
        return filePath;
    }

    public void setFilePath(String filePath) {
        this.filePath = filePath;
    }

    public String getFormat() {
        return format;
```

```
    }

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

    public long getTotalSize() {
        return totalSize;
    }

    public void setTotalSize(long totalSize) {
        this.totalSize = totalSize;
    }

    public LocalDateTime getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(LocalDateTime createdAt) {
        this.createdAt = createdAt;
    }

    public LocalDateTime getLastAccessedAt() {
        return lastAccessedAt;
    }

    public void setLastAccessedAt(LocalDateTime lastAccessedAt) {
        this.lastAccessedAt = lastAccessedAt;
    }


}
```

ArchivedFile

```
package model;

import java.time.LocalDateTime;

public class ArchivedFile extends ArchiveItem {
    private long originalSize;
    private long compressedSize;
    private String checksum;

    public ArchivedFile(String name, LocalDateTime modificationDate, long originalSize, long compressedSize, String
checksum) {
        super(name, modificationDate);
        this.originalSize = originalSize;
        this.compressedSize = compressedSize;
        this.checksum = checksum;
    }

    public long getOriginalSize() {
        return originalSize;
    }

    public void setOriginalSize(long originalSize) {
        this.originalSize = originalSize;
    }

    public long getCompressedSize() {
        return compressedSize;
    }
```

```
    public void setCompressedSize(long compressedSize) {
        this.compressedSize = compressedSize;
    }

    public String getChecksum() {
        return checksum;
    }

    public void setChecksum(String checksum) {
        this.checksum = checksum;
    }
}
```

ArchivedFolder

```
package model;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

public class ArchivedFolder extends ArchiveItem {
    private List<ArchiveItem> children = new ArrayList<>();

    public ArchivedFolder(String name, LocalDateTime modificationDate) {
        super(name, modificationDate);
    }

    public void addChild(ArchiveItem item) {
        children.add(item);
    }

    public List<ArchiveItem> getChildren() {
        return children;
    }
}
```

ArchiveFormat (enum)

```
package model;

public enum ArchiveFormat {
    ZIP, TAR_GZ, RAR, ACE;
    public static ArchiveFormat fromPath(String path) {
        String p = path.toLowerCase();
        if (p.endsWith(".zip")) return ZIP;
        if (p.endsWith(".tar.gz") || p.endsWith(".tgz")) return TAR_GZ;
        if (p.endsWith(".rar")) return RAR;
        if (p.endsWith(".ace")) return ACE;
        throw new IllegalArgumentException("Невідомий формат для файлу: " + path);
    }
}
```

ArchiveBookmark

```
package model;

import java.time.LocalDateTime;

public class ArchiveBookmark {
```

```
private long bookmarkID;
private long archiveID;
private String itemPath;
private String displayName;
private boolean isFolder;
private String note;
private LocalDateTime createdAt;

public ArchiveBookmark(long bookmarkID, long archiveID, String itemPath, String displayName,
                       boolean isFolder, String note, LocalDateTime createdAt) {
    this.bookmarkID = bookmarkID;
    this.archiveID = archiveID;
    this.itemPath = itemPath;
    this.displayName = displayName;
    this.isFolder = isFolder;
    this.note = note;
    this.createdAt = createdAt;
}

public long getBookmarkID() {
    return bookmarkID;
}

public void setBookmarkID(long bookmarkID) {
    this.bookmarkID = bookmarkID;
}

public long getArchiveID() {
    return archiveID;
}

public void setArchiveID(long archiveID) {
    this.archiveID = archiveID;
}

public String getItemPath() {
    return itemPath;
}

public void setItemPath(String itemPath) {
    this.itemPath = itemPath;
}

public String getDisplayName() {
    return displayName;
}

public void setDisplayName(String displayName) {
    this.displayName = displayName;
}

public boolean isFolder() {
    return isFolder;
}

public void setFolder(boolean folder) {
    isFolder = folder;
}

public String getNote() {
    return note;
}
public void setNote(String note) {
```

```
        this.note = note;
    }

    public LocalDateTime getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(LocalDateTime createdAt) {
        this.createdAt = createdAt;
    }
}
```

ArchiveItem

```
package model;

import java.time.LocalDateTime;

public abstract class ArchiveItem {
    protected String name;
    protected LocalDateTime modificationDate;

    public ArchiveItem(String name, LocalDateTime modificationDate) {
        this.name = name;
        this.modificationDate = modificationDate;
    }

    public String getName() {
        return name;
    }

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

    public LocalDateTime getModificationDate() {
        return modificationDate;
    }

    public void setModificationDate(LocalDateTime modificationDate) {
        this.modificationDate = modificationDate;
    }
}
```

OperationDetail

```
package model;


public class OperationDetail {
    private long detailID;
    private long logID;
    private String itemPath;
    private OperationType type;
    private OperationStatus status;
    private String message;

    public OperationDetail(long detailID, long logID, String itemPath, OperationType type,
                           OperationStatus status, String message) {
        this.detailID = detailID;
        this.logID = logID;
        this.itemPath = itemPath;
        this.type = type;
        this.status = status;
        this.message = message;
```

```
    }

    public long getDetailID() {
        return detailID;
    }

    public void setDetailID(long detailID) {
        this.detailID = detailID;
    }

    public long getLogID() {
        return logID;
    }

    public void setLogID(long logID) {
        this.logID = logID;
    }

    public String getItemPath() {
        return itemPath;
    }

    public void setItemPath(String itemPath) {
        this.itemPath = itemPath;
    }

    public OperationType getType() {
        return type;
    }

    public void setType(OperationType type) {
        this.type = type;
    }

    public OperationStatus getStatus() {
        return status;
    }

    public void setStatus(OperationStatus status) {
        this.status = status;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

OperationLog

```
package model;

import java.time.LocalDateTime;
public class OperationLog {
    private long id;
    private long archiveInfoId;
    private OperationType operationType;
    private OperationStatus status;
    private LocalDateTime timestamp;
    private String message;
```

```
public OperationLog(long id, long archiveInfoId, OperationType operationType,
            OperationStatus status, LocalDateTime timestamp, String message) {
    this.id = id;
    this.archiveInfoId = archiveInfoId;
    this.operationType = operationType;
    this.status = status;
    this.timestamp = timestamp;
    this.message = message;
}

public OperationLog(long archiveInfoId, OperationType operationType,
            OperationStatus status, String message) {
    this.id = 0;
    this.archiveInfoId = archiveInfoId;
    this.operationType = operationType;
    this.status = status;
    this.timestamp = LocalDateTime.now();
    this.message = message;
}

public long getId() {
    return id;
}

public void setId(long id) {
    this.id = id;
}

public long getArchiveInfoId() {
    return archiveInfoId;
}

public void setArchiveInfoId(long archiveInfoId) {
    this.archiveInfoId = archiveInfoId;
}

public OperationType getOperationType() {
    return operationType;
}

public void setOperationType(OperationType operationType) {
    this.operationType = operationType;
}

public OperationStatus getStatus() {
    return status;
}

public void setStatus(OperationStatus status) {
    this.status = status;
}

public LocalDateTime getTimestamp() {
    return timestamp;
}

public void setTimestamp(LocalDateTime timestamp) {
    this.timestamp = timestamp;
}

public String getMessage() {
    return message;
}
```

```
    public void setMessage(String message) {
        this.message = message;
    }
}
```

OperationStatus (enum)

```
package model;

public enum OperationStatus {
    SUCCESS, FAILURE, PARTIAL
}
```

OperationType (enum)

```
package model;

public enum OperationType {
CREATE, OPEN, EXTRACT, ADD, DELETE, TEST, CHECKSUM, SPLIT, JOIN
}
```

Пакет repository

IArchiveBookmarkRepository

```
package repository;

import model.ArchiveBookmark;
import java.util.List;

public interface IArchiveBookmarkRepository {
    void add(ArchiveBookmark bookmark);
    void delete(long bookmarkId);
    List<ArchiveBookmark> listByArchive(long archiveId);
}
```

IOperationDetailRepository

```
package repository;

import model.OperationDetail;
import java.util.List;

public interface IOperationDetailRepository {
    void add(OperationDetail detail);
    void addAll(List<OperationDetail> details);
    List<OperationDetail> listByOperation(long operationId);
}
```

ArchiveBookmarkRepositoryImpl

```
package repository;

import model.ArchiveBookmark;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

public class ArchiveBookmarkRepositoryImpl implements IArchiveBookmarkRepository {
    private final List<ArchiveBookmark> bookmarks = new ArrayList<>();
    private long nextId = 1;

    @Override
```

```
    public void add(ArchiveBookmark bookmark) {
        if (bookmark.getBookmarkID() == 0) {
            bookmark.setBookmarkID(nextId++);
        }
        bookmarks.add(bookmark);
        System.out.println("Додано закладку '" + bookmark.getDisplayName() + "'");
    }

    @Override
    public void delete(long bookmarkId) {
        bookmarks.removeIf(b -> b.getBookmarkID() == bookmarkId);
        System.out.println("Видалено закладку з ID: " + bookmarkId);
    }

    @Override
    public List<ArchiveBookmark> listByArchive(long archiveId) {
        System.out.println("Пошук закладок для архіву з ID: " + archiveId);
        return bookmarks.stream()
                .filter(b -> b.getArchiveID() == archiveId)
                .collect(Collectors.toList());
    }
}
```

OperationDetailRepositoryImpl

```
package repository;

import model.OperationDetail;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

public class OperationDetailRepositoryImpl implements IOperationDetailRepository {
    private final List<OperationDetail> details = new ArrayList<>();
    private long nextId = 1;

    @Override
    public void add(OperationDetail detail) {
        if (detail.getDetailID() == 0) {
            detail.setDetailID(nextId++);
        }
        details.add(detail);
    }

    @Override
    public void addAll(List<OperationDetail> detailList) {
        for (OperationDetail detail : detailList) {
            add(detail);
        }
        System.out.println("Додано " + detailList.size() + " деталей операції");
    }

    @Override
    public List<OperationDetail> listByOperation(long operationId) {
        System.out.println("Пошук деталей для операції з ID: " + operationId);
        return details.stream()
                .filter(d -> d.getLogID() == operationId)
                .collect(Collectors.toList());
    }
}
```

IArchiveInfoRepository

```
package repository;
import model.ArchiveInfo;
```

```
import java.util.List;
import java.util.Optional;

public interface IArchiveInfoRepository {
    Optional<ArchiveInfo> findByPath(String path);
    void upsert(ArchiveInfo info);
    List<ArchiveInfo> listRecent(int limit);
}
```

IOperationLogRepository

```
package repository;

import model.OperationLog;

import java.util.List;

public interface IOperationLogRepository {
    void add(OperationLog log);
    List<OperationLog> listByArchive(long archiveId, int limit);
}
```

ArchiveInfoRepositoryImpl

```
package repository;

import model.ArchiveInfo;
import repository.IArchiveInfoRepository;

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

public class ArchiveInfoRepositoryImpl implements IArchiveInfoRepository {
    private final List<ArchiveInfo> database = new ArrayList<>();

    @Override
    public Optional<ArchiveInfo> findByPath(String path) {
        return database.stream()
                .filter(info -> info.getFilePath().equals(path))
                .findFirst();
    }


    @Override
    public void upsert(ArchiveInfo info) {
        System.out.println("Імітація збереження/оновлення в БД для: " + info.getFilePath());
        Optional<ArchiveInfo> existing = findByPath(info.getFilePath());
        if (existing.isPresent()) {
            database.remove(existing.get());
        }
        database.add(info);
        System.out.println("Запис успішно збережено. В базі зараз " + database.size() + " записів.");
    }

    @Override
    public List<ArchiveInfo> listRecent(int limit) {
        if (database.size() <= limit) {
            return new ArrayList<>(database);
        }
```

```
        return new ArrayList<>(database.subList(database.size() - limit, database.size()));
    }
}
```

OperationLogRepositoryImpl

```
package repository;

import model.OperationLog;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;

public class OperationLogRepositoryImpl implements IOperationLogRepository {
    private final List<OperationLog> logs = new ArrayList<>();
    private long seq = 1;
    @Override
    public void add(OperationLog log) {
        if (log.getId() == 0) {
            log.setId(seq++);
        }
        logs.add(log);
    }

    @Override
    public List<OperationLog> listByArchive(long archiveId, int limit) {
        return logs.stream()
                .filter(l -> l.getArchiveInfoId() == archiveId)
                .sorted(Comparator.comparing(OperationLog::getTimestamp).reversed())
                .limit(limit)
                .collect(Collectors.toList());
    }
}
```

Пакет strategy

ArchiverStrategyFactory

```
package strategy;

import model.ArchiveFormat;

public class ArchiverStrategyFactory {

    private ArchiverStrategyFactory() {


    }

    public static IArchiverStrategy createStrategy(ArchiveFormat format) {
        switch (format) {
            case ZIP:
                return new ZipStrategy();
            case TAR_GZ:
            case RAR:
            case ACE:
            default:
                throw new UnsupportedOperationException("Формат ще не підтримується: " + format);
        }
```

```
    }

    public static IArchiverStrategy createForPath(String archivePath) {
        ArchiveFormat format = ArchiveFormat.fromPath(archivePath);
        return createStrategy(format);
    }
}
```

IArchiverStrategy

```
package strategy;

import model.Archive;
import java.util.List;
import java.util.Map;

public interface IArchiverStrategy {
    Archive create(String archivePath, List<String> files);
    Archive open(String archivePath);
    void extract(Archive archive, List<String> items, String destinationFolder);
    void add(Archive archive, List<String> files);
    void delete(Archive archive, List<String> itemPaths);
    boolean test(Archive archive);
    Map<String, String> checksum(Archive archive, List<String> itemPaths, String algorithm);
    List<String> split(Archive archive, int partSizeMB);
    String join(String firstPartPath);
}
```

ZipStrategy

```
package strategy;

import model.*;
import java.time.LocalDateTime;
import java.util.List;
import java.util.Map;
import java.util.HashMap;
import java.util.ArrayList;
import java.util.UUID;

public class ZipStrategy implements IArchiverStrategy {

    @Override
    public Archive create(String archivePath, List<String> files) {
        System.out.println("Створення ZIP архіву: " + archivePath);
        Archive archive = new Archive(archivePath, ArchiveFormat.ZIP);
        for (String file : files) {
            archive.getRootFolder().addChild(new ArchivedFile(file, LocalDateTime.now(), 1000, 500, "crc32-" +
UUID.randomUUID().toString().substring(0, 8)));
        }
        archive.setTotalSize(500L * files.size());
        return archive;
    }

    @Override
    public Archive open(String archivePath) {
        System.out.println("Відкриття ZIP архіву: " + archivePath);
        Archive archive = new Archive(archivePath, ArchiveFormat.ZIP);
        ArchivedFile file1 = new ArchivedFile("document.txt", LocalDateTime.now(), 1024, 450, "crc32-a1b2c3d4");
        archive.getRootFolder().addChild(file1);
        archive.setTotalSize(450);
```

```
        return archive;
    }

    @Override
    public void extract(Archive archive, List<String> items, String destinationFolder) {
        System.out.println("Розпакування з " + archive.getFilePath() + " до " + destinationFolder);
    }

    @Override
    public void add(Archive archive, List<String> files) {
        System.out.println("Додавання файлів до " + archive.getFilePath());
    }

    @Override
    public void delete(Archive archive, List<String> itemPaths) {
        System.out.println("Видалення з " + archive.getFilePath());
    }

    @Override
    public boolean test(Archive archive) {
        System.out.println("Тестування архіву " + archive.getFilePath());
        return true;
    }

    @Override
    public Map<String, String> checksum(Archive archive, List<String> itemPaths, String algorithm) {
        System.out.println("Розрахунок checksum для " + archive.getFilePath());
        return new HashMap<>();
    }

    @Override
    public List<String> split(Archive archive, int partSizeMB) {
        System.out.println("Розбиття на частини " + archive.getFilePath());
        return new ArrayList<>();
    }

    @Override
    public String join(String firstPartPath) {
        System.out.println("З'єднання частин, починаючи з " + firstPartPath);
        return firstPartPath.replace(".part1.zip", ".zip");
    }
}
```

Пакет facade

ArchiverFacade

```
package facade;
import model.*;
import repository.IArchiveBookmarkRepository;
import repository.IArchiveInfoRepository;
import repository.IOperationDetailRepository;
import repository.IOperationLogRepository;
import strategy.ArchiverStrategyFactory;
import strategy.IArchiverStrategy;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

public class ArchiverFacade {
    private final IArchiveInfoRepository archiveRepo;
    private final IOperationLogRepository logRepo;
```

```
    private Archive currentOpenArchive;
    private IArchiverStrategy currentStrategy;

    private final IArchiveBookmarkRepository bookmarkRepo;
    private final IOperationDetailRepository detailRepo;
    public ArchiverFacade(IArchiveInfoRepository archiveRepo, IOperationLogRepository logRepo,
                          IArchiveBookmarkRepository bookmarkRepo, IOperationDetailRepository detailRepo) {
        this.archiveRepo = archiveRepo;
        this.logRepo = logRepo;
        this.bookmarkRepo = bookmarkRepo;
        this.detailRepo = detailRepo;
    }

    public Archive create(String path, ArchiveFormat format, List<String> files) {
        try {
            currentStrategy = ArchiverStrategyFactory.createStrategy(format);
            currentOpenArchive = currentStrategy.create(path, files);
            ArchiveInfo newInfo = new ArchiveInfo(0L, path, format.name(), currentOpenArchive.getTotalSize(),
LocalDateTime.now(), LocalDateTime.now());
            archiveRepo.upsert(newInfo);
            log(OperationType.CREATE, OperationStatus.SUCCESS, "Створено архів", path);
            return currentOpenArchive;
        } catch (Exception e) {
            log(OperationType.CREATE, OperationStatus.FAILURE, e.getMessage(), path);
            throw new RuntimeException("Помилка створення архіву", e);
        }
    }

    public Archive open(String path) {
        try {
            currentStrategy = ArchiverStrategyFactory.createForPath(path);
            currentOpenArchive = currentStrategy.open(path);
            ArchiveInfo info = archiveRepo.findByPath(path)
                    .orElse(new ArchiveInfo(0L, path, currentOpenArchive.getFormat().name(),
currentOpenArchive.getTotalSize(), LocalDateTime.now(), LocalDateTime.now()));
            info.setLastAccessedAt(LocalDateTime.now());
            archiveRepo.upsert(info);
            log(OperationType.OPEN, OperationStatus.SUCCESS, "Відкрито архів", path);
            return currentOpenArchive;
        } catch (Exception e) {
            log(OperationType.OPEN, OperationStatus.FAILURE, e.getMessage(), path);
            throw new RuntimeException("Помилка відкриття архіву", e);
        }
    }

    public void addFilesToCurrentArchive(List<String> files) {
        ensureArchiveOpen();
        try {
            currentStrategy.add(currentOpenArchive, files);
            log(OperationType.ADD, OperationStatus.SUCCESS, "Додано файли", currentOpenArchive.getFilePath());
        } catch (Exception e) {
            log(OperationType.ADD, OperationStatus.FAILURE, e.getMessage(), currentOpenArchive.getFilePath());
            throw new RuntimeException(e);
        }
    }

    public void extractFromCurrentArchive(List<String> items, String destination) {
        ensureArchiveOpen();
        OperationLog logEntry = startLog(OperationType.EXTRACT);
        List<OperationDetail> details = new ArrayList<>();

        try {
            currentStrategy.extract(currentOpenArchive, items, destination);
```

```
            details.add(new OperationDetail(0, logEntry.getId(), "file1.txt", OperationType.EXTRACT,
OperationStatus.SUCCESS, "OK"));
            details.add(new OperationDetail(0, logEntry.getId(), "file2.txt", OperationType.EXTRACT,
OperationStatus.FAILURE, "Access denied"));

            finishLog(logEntry, OperationStatus.PARTIAL_SUCCESS, "Виконано розпакування з помилками", details);
        } catch (Exception e) {
            finishLog(logEntry, OperationStatus.FAILURE, e.getMessage(), null);
            throw new RuntimeException(e);
        }
    }
    public void deleteFromCurrentArchive(List<String> itemPaths) {
        ensureArchiveOpen();
        try {
            currentStrategy.delete(currentOpenArchive, itemPaths);
            log(OperationType.DELETE, OperationStatus.SUCCESS, "Видалено елементи",
currentOpenArchive.getFilePath());
        } catch (Exception e) {
            log(OperationType.DELETE, OperationStatus.FAILURE, e.getMessage(), currentOpenArchive.getFilePath());
            throw new RuntimeException(e);
        }
    }
    public boolean testCurrentArchive() {
        ensureArchiveOpen();
        try {
            boolean result = currentStrategy.test(currentOpenArchive);
            log(OperationType.TEST, OperationStatus.SUCCESS, "Тест завершено: " + (result ? "OK" : "Помилки"),
currentOpenArchive.getFilePath());
            return result;
        } catch (Exception e) {
            log(OperationType.TEST, OperationStatus.FAILURE, e.getMessage(), currentOpenArchive.getFilePath());
            throw new RuntimeException(e);
        }
    }

    public Map<String, String> checksumCurrentArchive(List<String> itemPaths, String algorithm) {
        ensureArchiveOpen();
        try {
            Map<String, String> checksums = currentStrategy.checksum(currentOpenArchive, itemPaths, algorithm);
            log(OperationType.CHECKSUM, OperationStatus.SUCCESS, "Розраховано контрольні суми",
currentOpenArchive.getFilePath());
            return checksums;
        } catch (Exception e) {
            log(OperationType.CHECKSUM, OperationStatus.FAILURE, e.getMessage(),
currentOpenArchive.getFilePath());
            throw new RuntimeException(e);
        }
    }
    public List<String> splitCurrentArchive(int partSizeMB) {
        ensureArchiveOpen();
        try {
            List<String> parts = currentStrategy.split(currentOpenArchive, partSizeMB);
            log(OperationType.SPLIT, OperationStatus.SUCCESS, "Архів розбито на частини",
currentOpenArchive.getFilePath());
            return parts;
        } catch (Exception e) {
            log(OperationType.SPLIT, OperationStatus.FAILURE, e.getMessage(), currentOpenArchive.getFilePath());
            throw new RuntimeException(e);
        }
    }

    public String joinArchiveFromParts(String firstPartPath) {
        try {
```

```
            IArchiverStrategy strategy = ArchiverStrategyFactory.createForPath(firstPartPath);
            String resultPath = strategy.join(firstPartPath);

            ArchiveFormat format = ArchiveFormat.fromPath(resultPath);
            ArchiveInfo newInfo = new ArchiveInfo(0L, resultPath, format.name(), 0L, LocalDateTime.now(),
LocalDateTime.now());
            archiveRepo.upsert(newInfo);

            log(OperationType.JOIN, OperationStatus.SUCCESS, "Частини успішно з'єднано в " + resultPath, resultPath);
            return resultPath;
        } catch (Exception e) {
            log(OperationType.JOIN, OperationStatus.FAILURE, e.getMessage(), firstPartPath);
            throw new RuntimeException(e);
        }
    }
    public void addBookmarkToCurrentArchive(String itemPath, String displayName, String note) {
        ensureArchiveOpen();
        long archiveId = archiveRepo.findByPath(currentOpenArchive.getFilePath())
                .map(archiveInfo -> archiveInfo.getId()).orElseThrow();

        ArchiveBookmark bookmark = new ArchiveBookmark(0, archiveId, itemPath, displayName, false, note,
LocalDateTime.now());
        bookmarkRepo.add(bookmark);
        System.out.println("Фасад: Додано закладку на " + itemPath);
    }

    public List<ArchiveBookmark> getBookmarksForCurrentArchive() {
        ensureArchiveOpen();
        long archiveId = archiveRepo.findByPath(currentOpenArchive.getFilePath())
                .map(archiveInfo -> archiveInfo.getId()).orElse(0L);
        return bookmarkRepo.listByArchive(archiveId);
    }

    public Archive getCurrentOpenArchive() {
        return currentOpenArchive;
    }

    public List<ArchiveInfo> listRecentArchives(int limit) {
        return archiveRepo.listRecent(limit);
    }
    public List<OperationLog> getOperationLogsForCurrentArchive(int limit) {
        ensureArchiveOpen();
        long archiveId = archiveRepo.findByPath(currentOpenArchive.getFilePath())
                .map(archiveInfo -> archiveInfo.getId()).orElse(0L);
        return logRepo.listByArchive(archiveId, limit);
    }

    private void log(OperationType type, OperationStatus status, String message, String archivePath) {
        long archiveId = archiveRepo.findByPath(archivePath)
                .map(archiveInfo -> archiveInfo.getId()).orElse(0L);
        logRepo.add(new OperationLog(archiveId, type, status, message));
    }

    private OperationLog startLog(OperationType type) {
        long archiveId = archiveRepo.findByPath(currentOpenArchive.getFilePath())
                .map(archiveInfo -> archiveInfo.getId()).orElse(0L);

        OperationLog logEntry = new OperationLog(archiveId, type, OperationStatus.PARTIAL, "In progress...");
        logRepo.add(logEntry);
        return logEntry;
    }

    private void finishLog(OperationLog logEntry, OperationStatus finalStatus, String message, List<OperationDetail>
```

```
details) {
        logEntry.setStatus(finalStatus);
        logEntry.setMessage(message);
        // logRepo.update(logEntry);

        if (details != null && !details.isEmpty()) {
            detailRepo.addAll(details);
        }
    }

    private void ensureArchiveOpen() {
        if (currentOpenArchive == null) {
            throw new IllegalStateException("Жоден архів не відкрито.");
        }
    }
}
```

Клас Main для демонстрації роботи

Main

```
import facade.ArchiverFacade;
import model.*;
import repository.*;

import java.util.List;

public class Main {
    public static void main(String[] args) {
        IArchiveInfoRepository archiveRepo = new ArchiveInfoRepositoryImpl();
        IOperationLogRepository logRepo = new OperationLogRepositoryImpl();
        IArchiveBookmarkRepository bookmarkRepo = new ArchiveBookmarkRepositoryImpl();
        IOperationDetailRepository detailRepo = new OperationDetailRepositoryImpl();
        ArchiverFacade facade = new ArchiverFacade(archiveRepo, logRepo, bookmarkRepo, detailRepo);


        System.out.println("Створення архіву");
        facade.create("C:\\docs.zip", ArchiveFormat.ZIP, List.of("file1.txt", "file2.txt"));

        System.out.println("\nВідкриття архіву");
        Archive opened = facade.open("C:\\docs.zip");
        System.out.println("Відкрито: " + opened.getFilePath());

        System.out.println("\nТестування архіву");
        boolean result = facade.testCurrentArchive();
        System.out.println("Результат тесту: " + result);
        facade.addFilesToCurrentArchive(List.of("new_file.txt"));
        System.out.println("\nВиконання операції з деталізацією");
        facade.extractFromCurrentArchive(List.of("file1.txt", "file2.txt"), "C:\\output");
        List<ArchiveInfo> recentArchives = facade.listRecentArchives(5);
        System.out.println("Список недавніх архівів:");
        for (ArchiveInfo info : recentArchives) {
            System.out.println(info.getFilePath() + " (останній доступ: " + info.getLastAccessedAt() + ")");
        }
        System.out.println("\nДодавання та перегляд закладок");
        facade.addBookmarkToCurrentArchive("file1.txt", "Важливий документ", "Перевірити перед відправкою");

        List<ArchiveBookmark> bookmarks = facade.getBookmarksForCurrentArchive();
        System.out.println("Закладки для поточного архіву (кількість): " + bookmarks.size());
        System.out.println("Історія операцій для поточного відкритого архіву (" +
facade.getCurrentOpenArchive().getFilePath() + "):");
        List<OperationLog> logs = facade.getOperationLogsForCurrentArchive(10);
```

```
        for (OperationLog log : logs) {
            System.out.println( log.getTimestamp() + " | " + log.getOperationType() + " | " + log.getStatus() + " | " +
log.getMessage());
        }
    }
}
```

Вивід

```
Створення архіву
Створення ZIP архіву: C:\docs.zip
Імітація збереження/оновлення в БД для: C:\docs.zip
Запис успішно збережено. В базі зараз 1 записів.

Відкриття архіву
Відкриття ZIP архіву: C:\docs.zip
Імітація збереження/оновлення в БД для: C:\docs.zip
Запис успішно збережено. В базі зараз 1 записів.
Відкрито: C:\docs.zip

Тестування архіву
Тестування архіву C:\docs.zip
Результат тесту: true
Додавання файлів до C:\docs.zip

Виконання операції з деталізацією
Розпакування з C:\docs.zip до C:\output
Додано 2 деталей операції
Список недавніх архівів:
C:\docs.zip (останній доступ: 2025-09-24T18:05:15.651739200)

Додавання та перегляд закладок
Додано закладку 'Важливий документ'
Фасад: Додано закладку на file1.txt
Пошук закладок для архіву з ID: 0
Закладки для поточного архіву (кількість): 1
Історія операцій для поточного відкритого архіву (C:\docs.zip):
2025-09-24T18:05:15.652739700 | TEST | SUCCESS | Тест завершено: OK
2025-09-24T18:05:15.652739700 | ADD | SUCCESS | Додано файли
2025-09-24T18:05:15.652739700 | EXTRACT | PARTIAL_SUCCESS | Виконано розпакування з помилками
2025-09-24T18:05:15.651739200 | CREATE | SUCCESS | Створено архів
2025-09-24T18:05:15.651739200 | OPEN | SUCCESS | Відкрито архів
```

Питання до лабораторної роботи

1. Що таке UML?

Це як креслення для програми. Уніфікована мова моделювання (UML) - це стандартний набір графічних позначень (діаграм), щоб візуально показати, як система побудована і як вона працює.

2. Що таке діаграма класів UML?

Це статичний "кресленик" коду. Вона показує, з яких "цеглинок" (класів) складається програма, які в них є дані (атрибути), що вони вміють робити (методи), і як ці класи пов'язані між собою.

3. Які діаграми UML називають канонічними?

Канонічними називають 8 основних діаграм, визначених у стандарті UML. Це:

Діаграма варіантів використання

Діаграма класів

Діаграма послідовності

Діаграма кооперації

Діаграма станів

Діаграма діяльності

Діаграма компонентів

Діаграма розгортання

4. Що таке діаграма варіантів використання?

Це діаграма, яка показує, ХТО (Актор) і ЩО (Варіант використання) може робити з системою. Вона не показує, як система це робить, а лише фіксує її основні функції з точки зору користувача. Для мого "Архіватора" вона показує, що Користувач може створювати, відкривати архіви і так далі.

5. Що таке варіант використання?

Це одна конкретна функція, яку система надає користувачу для досягнення якоїсь мети. Наприклад, "Створити архів" - це один варіант використання. "Додати файл" - інший.

6. Які відношення можуть бути відображені на діаграмі використання?

Основні чотири:

Асоціація: Простий зв'язок (Користувач ініціює "Створення архіву").

Включення (<<include>>): Одна дія обов'язково викликає іншу (у моїй роботі "Додати файл" включає "Відкрити архів").

Розширення (<<extend>>): Одна дія може доповнити іншу за певних умов (наприклад, "Оплатити замовлення" може бути розширено "Застосувати знижку").

Узагальнення: Відношення "є різновидом" (наприклад, актор "Адміністратор" є різновидом актора "Користувач").

7. Що таке сценарій?

Це детальний текстовий опис одного варіанта використання. Якщо варіант використання - це назва функції ("Створити архів"), то сценарій - це покрокова інструкція, як ця функція працює, включаючи всі можливі винятки.

8. Що таке діаграма класів?

Це головна діаграма для проектування, яка показує статичну структуру системи. Вона візуалізує класи, їхні дані (атрибути) та поведінку (методи), а також всі зв'язки між ними.

9. Які зв’язки між класами ви знаєте?

Я знаю такі основні зв'язки:

Асоціація: Класи "знають" один про одного.

Узагальнення (Наслідування): Відношення "є різновидом" (is-a).

Агрегація та Композиція: Відношення "є частиною" (has-a).

Залежність: Один клас "використовує" інший, але не зберігає посилання на нього постійно.

10. Чим відрізняється композиція від агрегації?

Обидва - це зв'язок "частина-ціле". Головна відмінність - у часі життя "частини".

Композиція: Жорсткий зв'язок. "Частина" не може існувати без "цілого". Аналогія: Будинок і кімнати. Якщо знести будинок, кімнати перестануть існувати.

Агрегація: Слабкий зв'язок. "Частина" може існувати сама по собі. Аналогія: Автомобіль і колеса. Якщо автомобіль розібрати, колеса залишаться.

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

Вони відрізняються графічно:

Композиція позначається зафарбованим (чорним) ромбом на стороні "цілого".

Агрегація позначається незафарбованим (білим) ромбом.

12. Що являють собою нормальні форми баз даних?

Це набір правил для проектування таблиць у базі даних. Головна мета - усунути дублювання даних та уникнути помилок при їх зміні. Зазвичай використовують перші три нормальні форми (1НФ, 2НФ, 3НФ).

13. Що таке фізична модель бази даних? Логічна?

Логічна модель - це те, як ми, програмісти, бачимо базу даних: у вигляді таблиць, колонок, ключів та зв'язків. Це наш "кресленик" даних.

Фізична модель - це те, як база даних реально зберігає ці дані на жорсткому диску: у вигляді файлів, блоків, індексів. Це вже внутрішня "кухня" самої СУБД.

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

Програмні класи - це об'єкти в пам'яті, а таблиці - це структури в базі даних. Щоб їх зв'язати, використовують підхід, який називається Object-Relational Mapping (ORM). У моїй роботі зв'язок простий: один клас-модель (ArchiveInfo, OperationLog) відповідає одній таблиці в базі даних. Це найпоширеніший варіант.

Висновки

Під час виконання цієї лабораторної роботи я розробив проект архітектури для програми "Архіватор". Головною метою було навчитися правильно планувати структуру програми ще до того, як писати основний код.

Спочатку я проаналізував, що має вміти програма, і намалював діаграму варіантів використання. Вона допомогла мені зрозуміти, які функції будуть у архіватора і як користувач буде з ними взаємодіяти. Потім я детально розписав три основні функції у вигляді сценаріїв, щоб уточнити їх логіку.

Далі я спроектував діаграму класів предметної області, щоб визначити основні кирпичики моєї програми: що таке "Архів", "Файл в архіві" і як вони пов'язані.

Найцікавішою частиною була розробка самої архітектури. Я використав кілька патернів проектування:

Repository, щоб відокремити роботу з базою даних від основної логіки.

Strategy та Factory, щоб програма могла легко працювати з різними форматами архівів (наприклад, ZIP) і в майбутньому можна було б просто додати підтримку RAR чи TAR.GZ.

Facade, щоб створити єдиний і простий "пульт керування" всією системою.

В результаті я отримав готовий скелет програми з продуманою структурою, діаграмами, що її описують, та базовим кодом.