

Міністерство освіти і науки України
Національний технічний університет України
„Київський політехнічний інститут ім. Ігоря Сікорського”
Кафедра інформаційних систем та технологій

Архіватор

Курсова робота

З дисципліни «ТЕХНОЛОГІЇ РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

Керівник ас. Вовк Є. А.
„Допущений до захисту”

(Особистий підпис керівника)

«___» _____ 2023 р.

Захищений з оцінкою

(оцінка)

«___» _____ 2023 р.

Члени комісії :

(Особистий підпис)

(Особистий підпис)

Виконавець

ст. Міщук М.Д.

зал. книжка № 1А-1417

(Особистий підпис виконавця)

«___» _____ 2023 р.

(Розшифровка підпису)

(Розшифровка підпису)

Київ – 2023

Національний технічний університет України
„Київський політехнічний інститут ім. Ігоря Сікорського”
Кафедра інформаційних систем та технологій
Дисципліна «Електроніка та мікропроцесорна техніка»

Курс 3 Група ІА-14 Семестр 5

ЗАВДАННЯ

на курсову роботу студента

Міщук Максим Дмитрович

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи): Архіватор
2. Термін здачі студентом закінченого проекту (роботи): 30.12.2023р.
3. Вихідні дані до роботи: предметна область «Архіватор»; шаблони проектування: strategy, adapter, factory method, facade, visitor; архітектура: р2р; загальні вимоги: Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, .rar, .ace, .7z) - додавання/ видалення файлів/папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці): огляд існуючих рішень, аналіз технічного завдання, опис моделі застосунку, розробка програмного забезпечення.
Додатки: Додаток А – опис моделі (UML-діаграми); Додаток Б – розробка програмного забезпечення (код програми); Додаток В – показ готового прототипу (скріншоти програми).
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): UML-діаграми (розгортання, прецедентів, діяльності, класу, послідовностей, станів) цілих систем та окремих компонентів, код програми, скріншоти програми.
6. Дата видачі завдання: 16.09.2023р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів виконання курсової роботи	Строк виконання етапів роботи	Примітки
1.	Видання теми курсової роботи	16.09.2023	
2.	Вступ, постановка задачі	20.09.2023	
3.	Огляд існуючих рішень	27.09.2023	
4.	Аналіз технічного завдання	03.10.2023	
5.	Опис моделі застосунку: Use-case, Activity та Class diagrams	05.10.2023	
6.	Опис моделі застосунку: Deployment, State та Sequence diagrams	13.10.2023	
7.	Розробка програмного забезпечення: Підготовчий етап	20.10.2023	
8.	Розробка програмного забезпечення: Pattern Strategy, Pattern Adapter	02.11.2023	
9.	Розробка програмного забезпечення: Pattern Factory Method, Pattern Facade	16.11.2023	
10.	Розробка програмного забезпечення: Pattern Visitor, Peer-to-Peer	02.12.2023	
11.	Розробка програмного забезпечення: Користувацький інтерфейс	16.12.2023	
12.	Титульний аркуш, завдання, висновок та список використаних джерел	23.12.2023	

Студент  (підпис) _____ (прізвище, ім'я, по батькові)

Керівник _____ (підпис) _____ (прізвище, ім'я, по батькові)

«_____» _____ 2023р.

ЗМІСТ

ВСТУП	5
1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ	6
2. АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ.....	11
2.1. Pattern Strategy	11
2.2. Pattern Adapter.....	12
2.3. Pattern Factory Method.....	13
2.4. Pattern Facade	14
2.5. Pattern Visitor	15
2.6. Peer-to-Peer	16
3. ОПИС МОДЕЛІ ЗАСТОСУНКУ	18
3.1. Deployment diagram.....	18
3.2. Class diagram	19
3.3. Use-case diagram	19
3.4. Activity diagram.....	20
3.5. State diagram.....	21
4. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	23
4.1. Підготовчий етап.....	23
4.2. Архіватор	25
4.3. Файлообмінювач	33
4.4. Реалізація користувацького інтерфейсу	34
ВИСНОВОК.....	40
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	42
ДОДАТКИ.....	43
Додаток А.....	43
Додаток Б	54
Додаток В	79

ВСТУП

Обмін інформацією є одним із найважливіших аспектів еволюції людства. Зробити відкриття – це лише один з етапів розвитку, ще потрібно щоб про нього дізналися інші. Нові покоління зможуть покращити твої досягнення та навіть використати їх для отримання власних винаходів. Можна тільки гадати скільки кардинально нових винаходів, технологій, теорем було втрачено через небажання або неможливість показувати їх суспільству.

На сьогоднішній день обмін даним тісно інтегрувалося в рутину нашого життя. Коли ми прокидаємося, то одразу беремо в руки телефон, щоб подивитися новини. Потрібно купити якусь електроніку – переглянули огляди, проаналізували історію цін після чого зробили замовлення. Більша частина проектів в індустрії інформаційних технологій є командними, тому для досягнення успіху одним з ключових моментів є вибір правильного інструменту для обміну даними.

Останнє й стало ключовою темою даної роботи. За мету поставлено розробити застосунок із використання отриманих знань з предмету «Технології розроблення програмного забезпечення», котрий дозволить зручно й швидко обмінюватися інформацією, а саме – файлами. Для того щоб зменшити навантаження на канали передачі, було також запропоновано вбудувати функціонал системи роботи з архівами, що дозволить транспортувати дані в більш компактному та зручному вигляді.

Розробка прототипу матиме в собі наступні кроки: огляд існуючих рішень, аналіз технічного завдання, огляд моделі майбутнього застосунку та сама розробка програмного забезпечення, поділена для зручності на розробку окремих компонентів системи.

1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

WinRAR – на даний момент є найпопулярнішою програмою для роботи з архівами. Існує на ринку вже 28 років та стабільно оновлюється. Підтримує широкий спектр форматів архівування та є єдиною програмою, спроможною створювати архіви формату RAR [1]. Додаток має потужний інструментарій та інтуїтивний інтерфейс, що у висновку й дає розуміння чому саме цей архіватор є найпопулярнішим.

Переваги:

- Постійна підтримка;
- Має потужний інструментарій: відновлення пошкоджених файлів, розділення архіву, встановлення паролів на архіви, підтримка drag-and-drop, створення саморозпаковувальних архівів;
- Застосунок є інтернаціоналізованим (підтримка більше 40 мов);
- Використання найкращих технологій стискання на ринку;
- Є безкоштовна «пробна» версія.

Недоліки:

- Має закритий код інструментарію для створення архівів типу RAR;
- Немає зворотної сумісності між форматами різного покоління типу RAR;
- Є платною.

7-Zip – це другий за популярністю архіватор. Також підтримує велику кількість різних типів архівування, включаючи свій формат стиснення 7z що використовує високоефективні алгоритми стиснення LZMA та LZMA2 [1]. Якщо порівнювати власні формати 7-Zip та WinRAR, то обидва є гарним вибором для використання, оскільки мають приблизно однакові результати при тестуванні.

Переваги:

- Підтримка великої кількості форматів;
- Власні технології стискання, що демонструють добрий результат;
- Застосунок є інтернаціоналізованим (підтримка 82 мов);
- Є абсолютно безкоштовним та має відкритий код.

Недоліки:

- В порівнянні з іншими архіваторами, інтерфейс менш інтуїтивно зрозумілий.

WinZip – теж досить відомий додаток для роботи з архівами. Зарекомендував себе як чудовий варіант для бізнес-проектів і для роботи в цілому. В основі лежить безпека та цільність, має власний інструментарій та інфраструктуру для використання хмарних середовищ та «шерінгу» файлів.

Переваги:

- Якісно зроблений інтуїтивний інтерфейс;
- Просунуті можливості з безпеки.

Недоліки:

- Немає безкоштовної версії.

PeaZip – є безкоштовною утилітою для роботи з архівами та має відкритий код. На відмінну від 7-Zip має доволі простий інтерфейс для використання, що є досить вагомою перевагою для тривіального користувача. Даний застосунок також узяв у себе функціонал з інших архіваторів: з WinZip додаткові функції шифрування та безпеки, з WinRAR можливість встановлювати паролі на архіви. Також цей інструмент стиснення може створювати, а також видобувати файли RAR. Якщо у вас встановлено WinRAR (навіть якщо пробна версія), PeaZip може скористатися програмним забезпеченням для повної підтримки RAR. [3]

Переваги:

- Простий та привабливий інтерфейс;
- Підтримка основних типів архівів, у тому числі можливість розпаковувати архіви типу RAR;
- Додатково є підтримка роботи зі знімками дисків формату ISO та UDF;
- Має додатковий функціонал платних аналогів.

Недоліки:

- Реалізація додаткового функціоналу реалізована більш складно ніж в інших архіваторах;
- Досить багато додаткових опцій при створенні та змінні архівів, що може здатися досить складним й незрозумілим звичайній людині, на початкових етапах.

Windows built-in archiver – починаючи з Windows ME, Microsoft інтегрувала підтримку архівів типу ZIP в операційну систему Windows [2]. Аналогом на операційній системі Mac OS є Archive Utility (стара назва BOMArchiveHelper, до Mac OS X 10.5), що з'явилась в версії Mac OS X 10.3 і так само має підтримку ZIP-архівів. Їх функціонал вплетений в роботу системи та якогось окремої програми для роботи з ними немає.

Переваги:

- Йдуть в комплекті з операційною системою;
- Інтерфейс інтегрований в файловий менеджер системи.

Недоліки:

- Підходять тільки для базових операцій розпакування та стиснення;
- Є доволі повільними в порівнянні з варіантами, описаними вище;
- Мають підтримку лише архівів типу ZIP.

Переглянувши та проаналізувавши вище оглянуті рішення було зроблено такі висновки:

- Система роботи з архівами буде розроблена з підтримкою найпопулярніших відкритих форматів, з можливістю: створення, редагування та видалення вмісту архівів.
- Архіватор буде мати відкритий код, та розділено на окремі компоненти (бібліотека архіватора може використовуватися окремо від користувацького інтерфейсу, таким чином можна буде легко інтегрувати її у власний код), що дозволить спільноті легше зрозуміти суть та самостійно підтримувати створений продукт.
- Основною аудиторією стануть: невеликі підприємства, що хочуть користуватися продуктом власного виробництва та яким потрібен лише базовий функціонал роботи з архівами.
- Також в сам архів буде влаштований файлообмінник, що дозволить швидко перекидати по мережі інформацію на пряму отримувачу без посередників.
- Інтерфейс буде примітивним та реалізований в консолі, але це не означає що він буде комплексний та не інтуїтивно зрозумілий. За мету поставлено зробити його простим для базового користувача, але при цьому витратити на це невелику кількість часу.

Встановивши мету та цілі, можна проаналізувати технічне завдання та розпочинати етап проектування майбутньої системи.

2. АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

Було поставлене технічне завдання розробити додаток з використання конкретних шаблонів та конкретної архітектури. Такі шаблони було задано:

- Strategy
- Adapter
- Factory Method
- Facade
- Visitor

Задана архітектура:

- Peer-to-Peer

Далі почнеться аналіз технічного завдання.

2.1. Pattern Strategy

Pattern Strategy – це поведінковий патерн проектування, який визначає сімейство схожих алгоритмів і розміщує кожен з них у власному класі. Після цього алгоритми можна замінювати один на інший прямо під час виконання програми. [4]

В застосунку є можливість створювати, редагувати, читати архіви. Але для кожного типу архівів треба свій підхід для роботи. Тому можна створити абстракції «Записник», «Читальник», «Видалятор», з відповідними абстрактними методами, а потім на основі них робити вже готові модулі, що будуть підставлятися відповідно до запиту користувача. Далі можна буде побачити приклад діаграми класів однієї з дій:

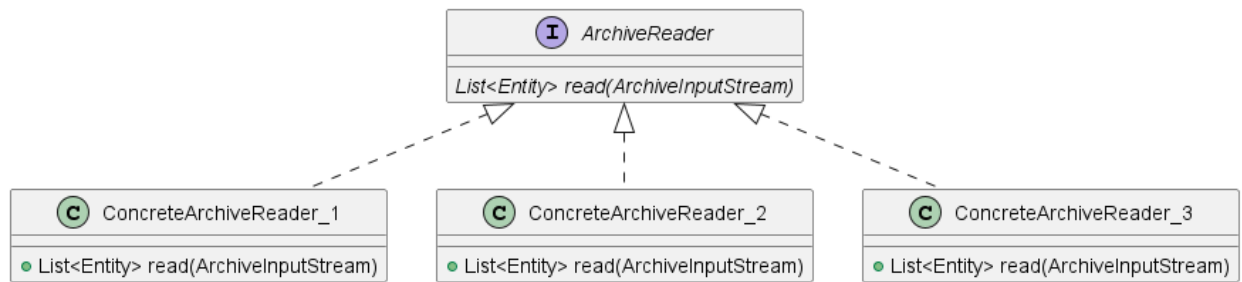


Рис. 2.1. Pattern Strategy

Методи приймають в себе потоки для роботи з архівами, котрі були заздалегідь створені (й продумано алгоритм закриття їх), що дозволяє сконцентруватися саме на алгоритмі виконання даних абстракцій.

Також слід пояснити логіку інтерфейсу *ArchiveWriter*. Метод *write(ArchiveOutputStream aos, List<Path> pathList)* – використовується для створення нового архіву (або заміщення вже існуючого), а метод *write(ArchiveInputStream ais, ArchiveOutputStream aos, List<Path> pathList)* – для додавання інформації вже в існуючий архів.

Логіка редагування архівів буде описана під час показу самої розробки програмного забезпечення.

2.2. Pattern Adapter

Pattern Adapter – це структурний патерн проектування, що дає змогу об'єктам із несумісними інтерфейсами працювати разом [4]. Суть полягає в створенні, обгортки, що спадкується від одного з двох несумісних компонентів, а приймає в себе протилежний компонент, тим самим створюється так званий «перехідник».

Так наприклад архіви типу TAR можуть бути додатково стиснуті різними компресорами, тоді перед тим як прийняти та відправити такий архів, його треба стиснути та розтиснути відповідно. За для того аби поведінки роботи з такими архівами не відрізнялася від інших, було створено адаптер що спадкується від звичайного архіву й приймає в себе компресор та базовий

архів та в середині робить з ними додаткові дії. Далі наведено діаграму класів цієї частини:

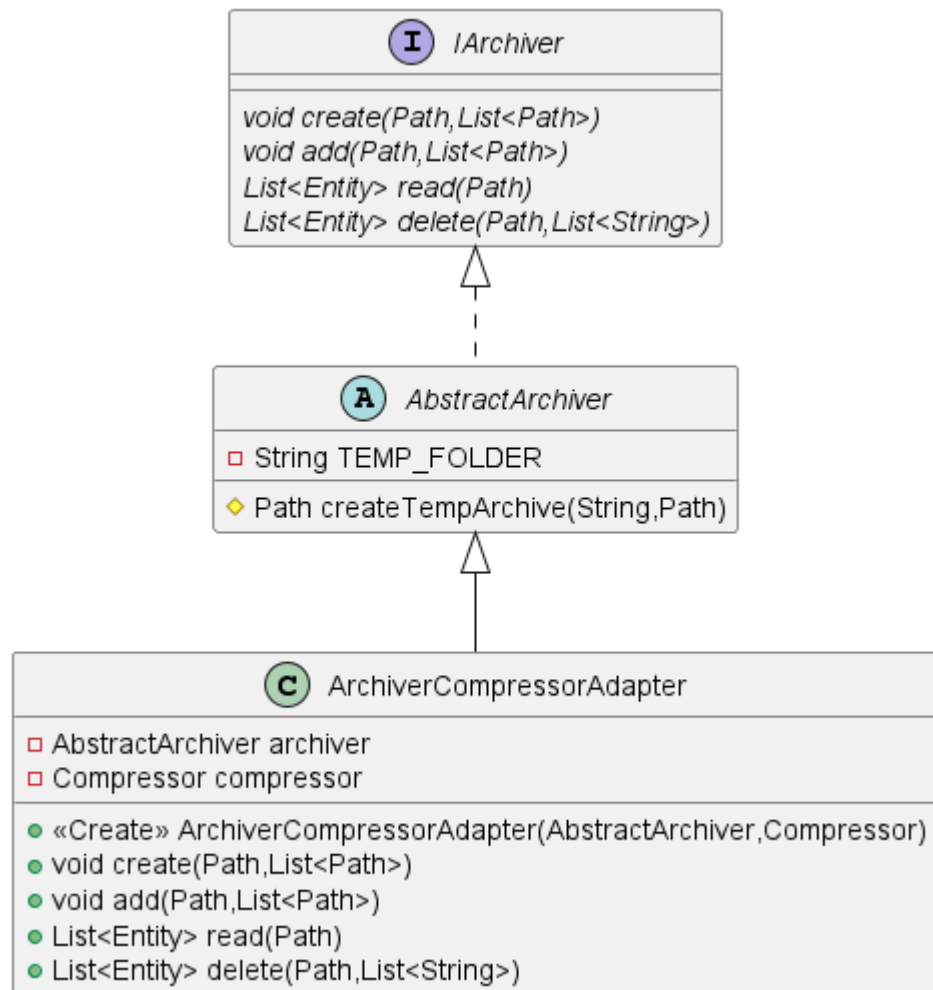


Рис. 2.2. Pattern Adapter

За створення тих чи інших типів архіватора, буде відповідати наступний шаблон, що дозволить зменшити написання інформації, для звичайних випадків користування.

2.3. Pattern Factory Method

Pattern Factory Method – це породжувальний патерн проектування, який визначає загальний інтерфейс для створення об’єктів у суперкласі, дозволяючи підкласам змінювати тип створюваних об’єктів. [4]

В проекті є вхідний клас архіватора, через який відбуваються усі взаємодії із застосунком, даний клас містить в собі посилання на чотири об’єкти дій

описаних вище в шаблоні стратегії. Так як працюючи з одним типом даних, то й всі дії будуть одного типу. Тому було створено фабричний метод, що приймає в себе тип архіву й повертає готовий до роботи клас. Відповідна діаграма класів:

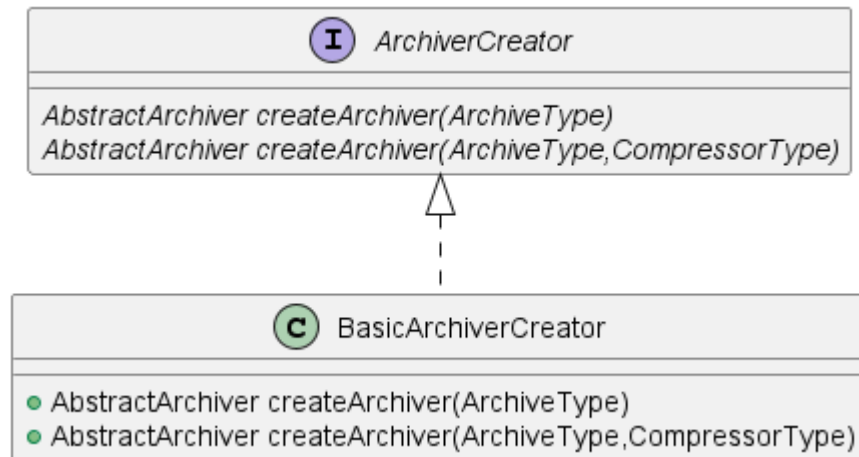


Рис. 2.3. Pattern Factory Method

Оскільки архіви можуть використовувати додаткове стиснення, то відбувається перевантаження методу створення на дві варіації: з додатковим стисненням, та без нього.

2.4. Pattern Facade

Pattern Facade – це структурний патерн проектування, який надає простий інтерфейс до складної системи класів, бібліотеки або фреймворку. [4]

Інтерфейсом взаємодії став клас архіватор, що приймає в себе тільки ту інформацію що саме потрібна від користувача, а подальші ініціалізації робить в середині без участі нього.

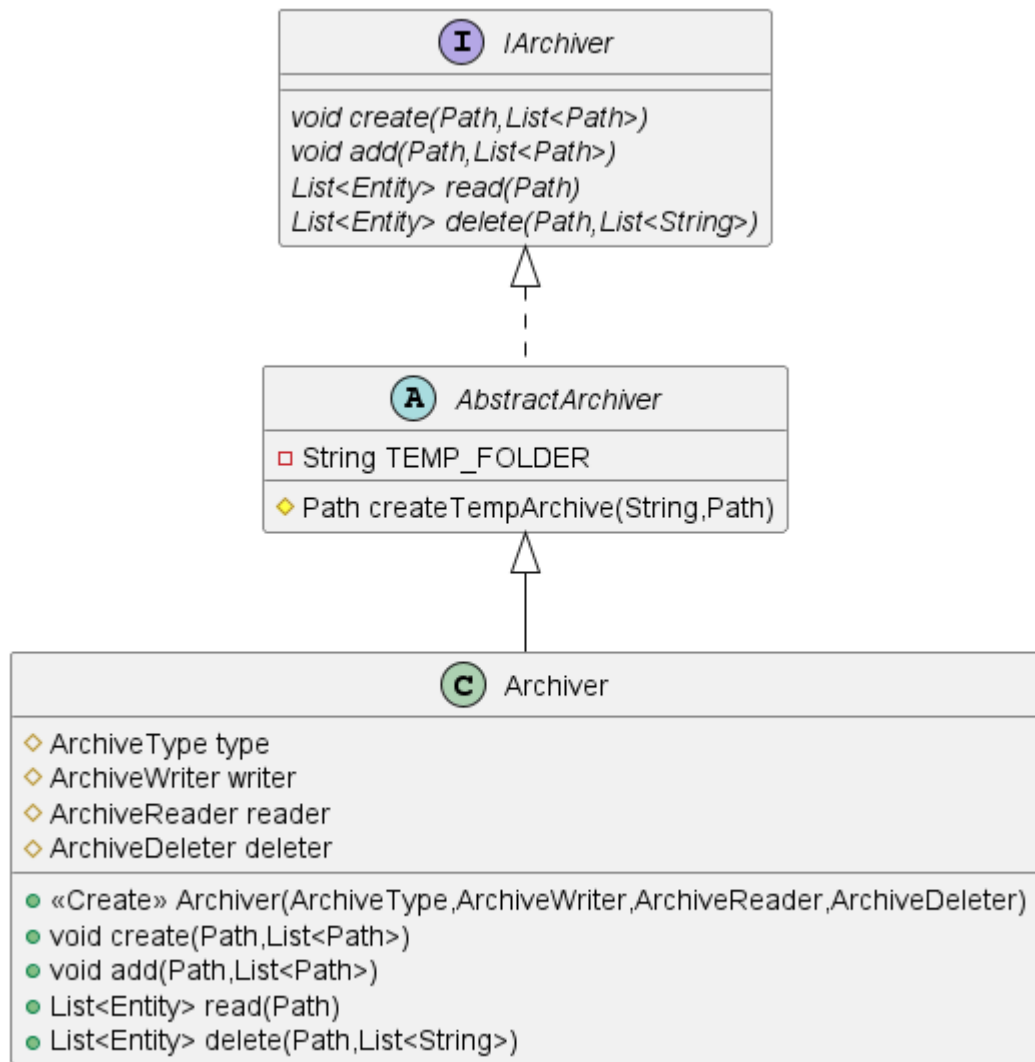


Рис. 2.4. Pattern Façade

Так, наприклад, якщо треба створити архів певного типу, то користувач надсилає тільки посилання для збереження та список файлів, що буде стиснуто.

2.5. Pattern Visitor

Pattern Visitor - це поведінковий патерн проектування, що дає змогу додавати до програми нові операції, не змінюючи класи об'єктів, над якими ці операції можуть виконуватися. [4]

На основі даного шаблону було зроблене розпакування файлів з архів. Кожна сутність архіву має в собі метод що приймає «відвідувача» і цей

відвідувач викликає свій функціонал над даною сутністю. Відвідувач зчитує дані сутності та записує їх за заданим шляхом користувачем.

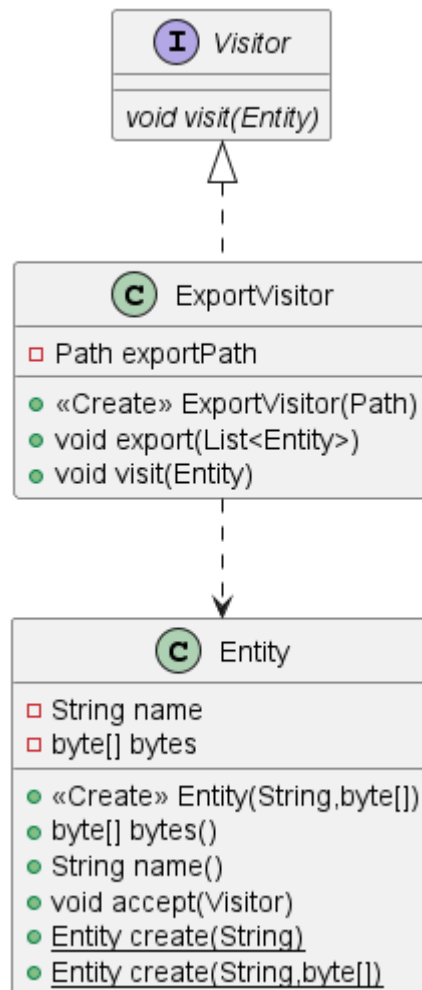


Рис. 2.5. Pattern Visitor

2.6. Peer-to-Peer

Архітектура peer-to-peer (p2p) - це модель комп'ютерної мережі, в якій кожен комп'ютер чи пристрій, що приєднаний до мережі, може виконувати роль і сервера, і клієнта одночасно. У цьому типі мережі відсутні централізовані сервери, котрі контролюють та керують комунікацією між вузлами. Замість цього, вузли взаємодіють напряму один з одним. [5]

На основі даної архітектури було розроблено файлообмінювач, що встановлюється на декілька комп'ютерів та резервує під себе один порт й дає змогу за IP-адресою встановлювати з'єднання та відправляти файли на інші

клієнти. Кожин користувач може як приймати та надсилати файли, головне знати адресу за якою відправляти. Також користувачі можуть самі обирати коли вмикати й вимикати їх, за для потреби й економії ресурсів системи.

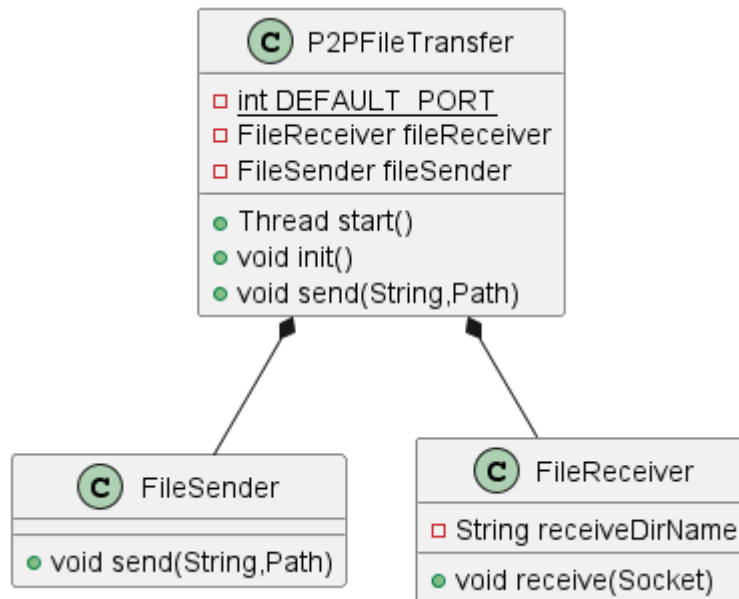


Рис. 2.6. Peer-to-Peer

Метод *init()* відповідає за запуск серверної частини клієнта. Коли з'єднання з клієнтом встановлено, дані передаються класу *FileReceiver*, що вже обробляє запит. Дані будуть зберігатися в папці отриманих повідомлень. Метод *send()* – за клієнтську частину, першим параметром поступає адреса, а другим файл для передачі. Дані так само передаються відповідному класу *FileSender*, що за заданим файлом дістане його мета-дані й основні дані, та надсилає їх на серверну частину іншого клієнта.

Також сумісне використання файлообмінювача та архіватора дозволить надсилати великі об'єми файлів з стиснутій формі, що зменшить навантаження на мережу в цілому.

3. ОПИС МОДЕЛІ ЗАСТОСУНКУ

Проаналізувавши вхідні значення, можна перейти до розробки UML-діаграм, за для полегшення розуміння проекту в цілому. Було використано як і структурні діаграми так і поведінкові. Загалом було розроблено шість діаграм:

Структурні:

- Deployment diagram
- Class diagram

Поведінкові:

- Use-case diagram
- Activity diagram
- State diagram
- Sequence diagram

Використання стандартизованого підходу дозволило вирішити більшу кількість питань та проблем ще на стадії проектування моделі та пришвидшило написання застосунку в цілому.

3.1. Deployment diagram

Діаграма розгортання – цей вид UML-діаграм використовується для моделювання фізичної структури системи, розташування програмного забезпечення та його компонентів на відповідних апаратних компонентах системи. [6]

Даний тип діаграм буває двох видів: екземплярний та описовий. Різниця полягає в кількості продемонстрованої інформації. Останній тип використовується на ранніх етапах розробки для основного опису як розгортається система на фізичному пристрої, тому було обрано саме її (*Додаток А – Deployment diagram*).

З діаграми можна побачити що застосунок (середовище виконання) буде складатися з двох частин (артефактів): один відповідає за функціонал роботи з архівами, а інший за обмін файлами між пристроями. Увесь застосунок в буде взаємодіяти з іншим середовищем виконання – файловою системою, для зчитування та запису даних. Сам обмін між пристроями буде відбуватися за протоколом TCP/IP, тобто з попереднім встановленням з'єднання.

На діаграмі показано тільки два пристрої, хоча в реальному середовищі їх може бути значно більше.

3.2. Class diagram

Одним з технічних завдань було використання об'єктно-орієнтованого стилю програмування, тому для правильного розуміння структури була і розроблена діаграма класів.

Діаграма класів показує статичну структуру системи в цілому, включаючи класи, їх атрибути й поведінку та зв'язки з іншими класами.[6] Готову діаграму показано в *Додатку А – Class diagram*.

З діаграми можна побачити що більшу частина компонентів побудована на абстракціях у вигляді інтерфейсів, а деякі на абстрактних класах, що в свою чергу побудовані на інтерфейсах. Інтерфейси дали змогу задекларувати основні методи взаємодію з конкретними нащадками, а абстрактні класи – додати певний загальний функціонал, поля та логіку.

Подальші рішення з приводу реалізації тих чи інших класів (наприклад, клас *AbstractArchiver*) буде описано в наступних розділах безпосередньо під час розробки.

3.3. Use-case diagram

Діаграма прецедентів демонструє способи взаємодії користувача (в даному типі діаграм це актори) з розробленою системою. Цей вид діаграм є високорівневим та більш-концептуальний, щоб описати вимоги до

створюваної системи. [6] Таким чином діаграма використання має наступний вид (*Додаток А – Use-case diagram*).

Зверху було продемонстровано діаграму використання для системи архіватора. Через те, що по взаємодії система є доволі простою, актор тут один. Також слід зазначити що для спрощення розуміння, деякі дії було узагальнено та винесено на окремі частини діаграми.

Так можна побачити що редагування архіву може бути виконано як додаванням об'єкту до архіву, так і видаленням іншого. Далі буде продемонстрована діаграма використання для файлообмінювача.

Ця система теж є доволі простою й включає в себе лише один вид взаємодії – надсилання файлу. Однак ця дія включає в себе ще дві дії: перша – надіслати запит на посилання (відправник), друга – прийняти запит (отримувач).

3.4. Activity diagram

Діаграму діяльності можна назвати продовженням діаграми прецедентів. Вона візуалізує хід виконання операцій в компонентах системи. [7] Основна ціль: показати алгоритм виконання певного процесу: розгалуження, розбиття на паралельні потоки виконання, зациклення та переривання процесу. [6] В *Додатку А – Activity diagram* наведено діаграму діяльності, для процесу створення нового архіву.

З отриманої діаграми можна побачити нову сутність – користувачький інтерфейс та базовий опис його логіки. Користувач задає всю необхідну інформацію для створення нового екземпляру, ці дані перевіряються на стороні користувача (наприклад, чи посилання на файл існує), після чого відправляються на сторону архіватора. Він теж перевіряє запит на коректність, але в даному випадку вже перериває виконання блок-схеми. Взагалі, якщо узагальнити усі типи взаємодії в системі архіватора, то можна їх привести до

наступного вигляду, що знаходиться в *Додатку А – Activity diagram (Archive Basic)*.

Користувач надсилає запит на дію, отримує відповідну форму, заповнює її, перевіряється коректність на стороні інтерфейсу, відправляється запит до системи архіватора.

Для файлообмінювача діаграми діяльності мають де що схоже та відмінне одночасно. Діаграма відправлення файлу на серверну частину іншого клієнта є в *Додатку А – Activity diagram (File Transfer – send)*.

3.5. State diagram

Діаграма станів показує стан як окремих компонентів, так і системи в цілому. Між станами існують перехідні процеси, що явно показують як система перейшла від одного стану, до іншого [6]. Самі стани теж можуть мати в середині свої стани та процеси, що може сильно покращити розуміння поведінки системи, без знання деталей виконання.

Цей тип діаграм ідеально підходить для проектування системи логування під час розробки застосунку, бо явно показує які стани має система. Діаграма описана для однієї з частин моделі – додавання нових файлів до архіву та знаходиться в *Додатку А – State diagram (Archive Adding)*.

На описаному прикладі можна побачити діаграму станів під час додавання інформації до вже існуючого архіву. Архіватор отримує запит на додавання, перевіряє його, після чого відбувається розгалуження: якщо запит коректний, то відбувається перехід до безпосередньо додавання інформації в архів, а якщо ні – до виключної ситуації. Після успішного додавання, відбувається перебіг до стану успішного виконання, після чого процес закінчується.

3.6. Sequence diagram

Діаграма послідовності за мету ставить показати опис системи в динаміці, час життя її сутностей, виходячи з діаграми класів. На ній показується як саме екземпляри класів взаємодіють між собою, які методи викликають, які результати отримують в залежності від тих, чи інших подій [7]. Відображення цієї діаграми буде показано в *Додатку А – Sequence diagram (Archive Creation)*.

Тут можна побачити послідовність створення нового архіву на стороні клієнта, від самого запуску застосунку. Напочатку користувач знаходиться в головному середовищі, після чого відбувається перехід в середовище архіватора, де вже викликає команду створення нового архіву. Результатом виконання даної команди є отримання форми, котру треба заповнити. Після заповнення якої, відбувається формування запиту та відправлення його на сторону архіватора.

На стороні системи роботи з архівами відбувається прийняття запиту, після чого відбувається перевірка вхідних даних. Тут виникає розгалуження: або запит йде далі по схемі, або повертається помилка, у разі виникнення виключної ситуації. Сам результат (незалежно від успішності виконання) повертається назад до користувацького інтерфейсу.

Після того як всі діаграми було затверджено та всі проблеми на проектувальному етапі було вирішено, можна безпосередньо перейти до етапу розробки даної системи.

4. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Підготовчий етап

Для написання системи буде використовуватись мова програмування Java, версії JDK 17 (LTS). Даний вибір був зроблений по декільком причинам, але основною є колишній досвід роботи з цією мовою. Основні переваги Java:

- Портативність. Код, написаний на Java можна запустити на будь-якій системі, що підтримує JVM (віртуальна машина Java). Це в свою чергу дозволить одразу тестувати прототипи застосунку на майже будь-якій звичайній операційній системі.
- ООП. Все побудовано на об'єктно-орієнтованому програмуванні, що ідеально підходить під задані умови технічного завдання.
- Зворотня сумісність. Java відома своїм принципом що до зворотної сумісності, тому якщо навіть через 5 років, розроблений застосунок буде підтримуватися новими версіями Java, але при цьому може ще й нарощувати новий функціонал
- Потужність влаштованих бібліотек. Java «з коробки» має дуже широкий функціонал, наприклад: багатопоточність та функціональний стиль програмування.
- Garbage Collector. Наявність можливості автоматично керувати пам'яттю, позбавляє розробника зайвої роботи й пришвидшує розробку застосунку й взагалі системи

Також влаштовані бібліотеки мають в Java мають функціонал для роботи тільки з архівами типу ZIP та JAR, тому для полегшення розроблення буде використовуватися бібліотека Apache Commons Compress.

Apache Commons Compress – бібліотека що визначає API для роботи з архівами та стисненнями типу: ar, cpio, Unix dump, tar, zip, gzip, XZ, Pack200, bzip2, 7z, arj, LZMA, snappy, DEFLATE, lz4, Brotli, Zstandard, DEFLATE64 та Z files. [4]

Робота з даним прикладним програмним інтерфейсом нічим не відрізняється від влаштованих в саму Java потоків для роботи з архівами, але для роботи з архівами типу ZIP і JAR, він використовує власні реалізації цих потоків.

Всі потоки даної бібліотеки спадкуються від абстрактних класів *ArchiveOutputStream* та *ArchiveInputStream* (для виведення та введення відповідно). Для створення потоку треба створити файловий потік до архіву та передати його в параметр конструктора відповідного типу архіву. Також для полегшення створення потоків, в бібліотеці було передбачено інтерфейс *ArchiveStreamProvider* та *ArchiveStreamFactory*, що разом утворюють абстрактну фабрику. Саме цей спосіб буде використано під розробки.

Далі треба продемонструвати алгоритм взаємодії з архівами. Архіви складаються з записів (*ArchiveEntry*), тому для взаємодії треба спочатку їх відкрити/створити (*getNextEntry()/createArchiveEntry()*) після чого закрити (*closeArchiveEntry()*) (Рис. 4.1.).

```
try (
    ArchiveOutputStream aos = new ZipArchiveOutputStream(Path.of("test.zip"));
    BufferedInputStream bis = new BufferedInputStream(new FileInputStream("test.txt"))
) {
    ArchiveEntry newEntry = aos.createArchiveEntry(Path.of("test.txt"), "test.txt");
    aos.putArchiveEntry(newEntry);
    IOUtils.copy(bis, aos);
    aos.closeArchiveEntry();
} catch (IOException ex) {
    ex.printStackTrace();
}
```

Рис. 4.1. Приклад використання Apache Commons Compress

Також для ведення логування буде використовуватися фреймворк *Log4j* версії 2.x.x.. Виведення відбуватиметься в папку *logs* у відповідний розділ та відповідний рівень логування (в залежності від рівню логування та частини застосунку). Файл конфігурації наведено в Додатку Б – *log4j2.xml*.

Функціонал файлообмінювача буде реалізований за допомоги *Socket* та *ServerSocket*. Сокет — це абстрактне поняття, що визначає можливість для

програми встановити з'єднання для обміну даними в мережі. Конструктор об'єкта сокету приймає адресу віддаленого сервера і номер порту. [5]

Слід зазначити що максимальний розмір одного блоку передачі по сокету не повинен перебільшувати 64 кілобайти, а навіть якщо файл буде менше заданого розміру, то при пересиланні декількох файлів, треба розуміти де починається та закінчується певний файл. Також якщо передавати виключно інформації що знаходиться в середині, то можна втратити основні дані файлу, як приклад назву та розширення файлу. [6]

Для вирішення першої проблеми буде використано *DataInputStream* та *DataOutputStream*, куди буде в певній послідовності записуватися інформація, котру можна легко зчитати в такій самій послідовності. Послідовність задана така: ім'я файлу (включаючи розширення), дата створення (*long*), дата останньої зміни (*long*), розмір файлу (*long*), та сама послідовність байтів. Через те що програма буде знати розмір файлу, то вона зчитає саму ту кількість байтів, яка належить саме цьому файлу. А проблему обмеженого розміру блоку при передачі, було вирішено розділенням файлів на чанки фіксованого розміру – 4 кілобайти. [6]

Також, щоб коли клієнт працює як сервер не «замерзав» увесь застосунок під час очікування запиту, було прийнято рішення використовувати *Thread* в середині якого й запускатиметься сервер. Це дозволить відокремити роботу клієнту файлообмінювача, від інших частин програми. Далі перейдемо безпосередньо до реалізації компонентів систем.

4.2. Архіватор

4.2.1. Реалізація дій архіватора

Частина відповідальна з взаємодію з архівами знаходиться в пакеті *archiver_api.actions*. Всередині цього пакету під кожную дію виділений окремий пакет: *write*, *read*, *delete*. Всередині цих пакетів лежать інтерфейси, що

делкарують методи для взаємодії, та пакети *impl*, де знаходяться реалізації методів відповідних інтерфейсів. Далі буде оглянуто детально кожну дію.

4.2.1.1. Створення та додавання

За виконання цих двох функцій відповідає пакет `archiver_api.actions.write`, де описано два методи запису даних в архів (*Додаток Б – ArchiveWriter.java*).

Перший метод використовується для створення нового архіву, та приймає в себе потік виводу архіву, де буде ініціалізація архіву, та список шляхів до файлів, що буде додано в архів.

Другий метод використовується для додання інформації до вже існуючого архіву. Він приймає в себе потік вводу архіву, потік виводу архіву, та список шляхів файлів, да додавання в архів.

Так як запис даних до архіву нічим не відрізняється для різних типів, то було розроблено один клас, реалізує методи задекларовані в інтерфейсі `ArchiveWriter`. Реалізація знаходиться в *Додатку Б – BasicArchiveWriter.java*.

Перший метод:

Всередині знаходиться цикл, що проходиться по всім переданим шляхам для збереження. Він перевіряє чи є це файлом, чи директорією, якщо цей шлях є директорією то перебрає всі файли що знаходяться всередині цієї директорії (включаючи внутрішні директорії), та встановлює відносний шлях починаючи з «точки входу». Таким чином архів може являти собою не просто перелік файлів, а вже повноцінну ієрархію.

Другий метод:

Даний алгоритм розпочинається з формування списку імен файлів для додавання. Потім відбувається перевірка, чи є додана назва у вже існуючому архіві. Якщо архів буде містити файли з однаковою назвою, то новий файл буде перезаписувати раніше створений.

Варто зазначити що зміни робляться не в існуючому архіві, а на основі заданих параметрів формується новий архів, що після успішного створення заміщує старий.

4.2.1.2. Читання

В даній функції за мету поставлено можливість перегляду архіву у двох форматах: основна інформація та детальна. Таким чином можна якщо треба просто переглянути вміст архіву, то до користувача прийде лише основна інформація, що є більш раціональною витратою ресурсів. За дану дію відповідає пакет *archiver_api.actions.read*.

Почнемо з основного інтерфейсу (*Додаток Б – ArchiveReader.java*). Даний інтерфейс має в відповідно два методи: базового та повного читання. На вхід в обох випадках поступає потоку вводу архіву, а на вихід йду параметризований список типу сутностей, ще несуть в собі інформацію про об'єкти, збережені у вхідному архіві (*Додаток Б – Entity.java*).

Слід зазначити що *Entity* є універсальним як і для виводу в користувацький інтерфейс, так і для розпакування архівів. Для створення того чи іншого варіанту передбачено відповідні статичні методи *create()*: з ініціалізацією всіх полів, чи тільки з полем назви сутності. Приклад використання детального варіанту буде показаний через два підрозділи. Реалізацію інтерфейсу читання наведено в *Додатку Б – BasicArchiveReader.java*:

Читання являє собою прохід по всім записам в архіві, з формування вихідних сутностей в залежності від обраного методу. Після чого готовий список даних передається нагору.

4.2.1.3. Видалення

В даній частині реалізовано функціонал видалення окремих записів (або групи записів) з архіву. Функціонал описаний за даним шляхом:

archiver_api.actions.delete. Основний інтерфейс представлений в Додатку Б – *ArchiveDeleter.java*.

В інтерфейсі задекларовано лише один метод видалення. Параметрами являються потоки вводу та виводу архіву, та список імен сутностей для видалення. За для формування зворотного зв'язку на вихід поступає список базової інформації про сутності, що було видалено. Цей список можна буде відобразити у результаті видалення на стороні користувацького інтерфейсу. Реалізації логіки лежить у Додатку Б – *BasicArchiveDeleter.java*:

BasicArchiveDeleter.java

Під час видалення створюється новий архів куди записуються усі об'єкти чий назви не було в списку видалення, а інформація інших записів поступає до списку видалених сутностей. Таким чином, після видалення, користувач отримує список, а новостворений архів заміщує старий (механізм заміщення буде описаний нижче).

Наступним кроком буде реалізація вхідного інтерфейсу, що буде посилатися на вищеописані дії.

4.2.2. Реалізація вхідного інтерфейсу

Функціонал даної частини знаходиться в пакеті *archive_api.archivers*. На поверхні знаходяться абстракції, що декларують функціонал, а в пакеті *impl* вже знаходяться конкретні реалізації. Опис самого вхідного інтерфейсу знаходиться в Додатку Б – *IArchiver.java*.

Даний інтерфейс можна назвати вхідною точкою до системи роботи з архівами, він приймає тільки ті дані, що безпосередньо потребують введення від самого користувача. Якщо користувач захоче використовувати напряму функціонал, описаний в попередньому підрозділі, то він повинен сам проініціалізувати потоки для роботи, та написати логіку заміщення архівів під час їх видозміни. Саме для цього і було створено даний інтерфейс, котрий в середині виконує те, про що звичайний користувач не хотів би задумуватися.

Також даний підхід дозволить зменшити ймовірність появи виключних ситуацій, що зробить загальний стан системи стабільніше.

Інтерфейс складається з чотирьох методів, що посилаються на відповідні методи інструментів, описаних в минулому підрозділі. На входи приходять лише шляхи й назви до архівів та файлів.

Варто звернути увагу що метод читання архіву приймає в себе значення типу *ArchiveReadingType* (Рис. 4.2.), на основі якого обирається метод читання. Базова частина логіки архіватора знаходиться в Додатку Б – *AbstractArchiver.java*.

```
public enum ArchiveReadingType {
    3 usages
    BASIC, FULL
}
```

Рис. 4.2. Способи читання архіву

Даний клас є абстрактним, та в ньому реалізована логіка ініціалізації тимчасової папки та архівів у ній. Таким чином усі видозміни відбуваються не напряду в архіві, а в її тимчасовій копії. Це зроблено в цілях уберегти оригінал від пошкоджень, у разі викидані виключень. Метод створення тимчасового архіву повертає відповідно шлях до цього архіву.

Тепер переходимо безпосередньо до реалізації «фасаду» (Додаток Б – *Archiver.java*).

Даний клас лежить в пакеті *archiver_api.archivers.impl*, та описує в собі повну логіку інтерфейсів наведених вище. Даний клас містить в собі чотири поля: три з них посилаються на інтерфейси з пакету *archiver_api.actions*, а четвертий вказує тип архіву, з яким працює даний архіватор. Останній тип обирається з переліку *ArchiveType*, що лежить в пакеті *archiver_api.supported_types*. Усі чотири поля повинні бути вказаними в параметрах конструктора під час створення відповідного архіватору.

В основі кожної дії лежить створення відповідний потоків на основі заданих параметрів. Самі потоки створюються у конструкції *try-with-resources*, що гарантується закриття потоків після їх виконання у незалежності від результату виконання.

У методів що виконують дії модифікації та видалення, напочатку створюється тимчасовий архів, з яким будуть відбуватися подальші дії. Після успішного завершення операції, відбувається перенесення нового архіву до оригінальної директорії із заміною минулого варіанту. Блок *final* відповідає за видалення тимчасового файлу, в незалежності від вихідного результату програми.

Також для облегшення створення архіватора, було створено інтерфейс за шаблоном фабричного методу, що знаходиться в пакеті *archive_api.archivers*. Він містить метод *createArchiver()*, котрий приймає в себе тип з яким буде працювати архіватор та повертає сутність типу *AbstractArchiver* (Додаток Б – *ArchiveCreator.java*).

Як можна побачити даний метод є перевантаженим та може в себе приймати окрім типу самого архіву, ще тип компресора. Про реалізацію й логіку другого варіанту буде показано в наступному підрозділі.

Даний інтерфейс імплементує клас *BasicArchiveCreator*. Як зрозуміло з назви він буде створювати систему роботи з архівами, збудовану на основі базових реалізацій пакету *archiver_api.actions*. Використання даного підходу полегшить створення архіваторів майбутнім користувачам бібліотеки, а також зменшить кількість коду, необхідну для створення архіватора.

4.2.3. Реалізація додаткового стиснення

Алгоритм роботи компресора доволі простий: перед початком роботи зі стиснутим архівом треба його розтиснути й у кінці назад стиснути. Класи для роботи з цим типом лежать в пакеті *archiver_api.compressors*. Починаємо з інтерфейсу (Додаток Б – *Icompressor.java*).

Інтерфейс складається з двох методів: стиснення й розтиснення. Першим параметром вони приймають в себе шлях до існуючого архіву, а другим – шлях до збереження результату. Показано клас, що описує їх логіку в *Додатку Б – Compressor.java*:

Суть методу стиснення полягає в передачі даних потоку вводу архіву до потоку виводу компресора. А методу розтиснення – в зчитуванні даних з потоку вводу компресора й передачі їх у потік виводу архіву.

За для запобігання дублювання коду, й збереження логіки та поведінки застосунку було розроблено клас за шаблоном адаптер, ціль якого зробити зв'язок можливість взаємодіяти із стиснутими архівами, як із звичайними. Даний адаптер знаходиться в пакеті *archiver_api.archivers.impl* та реалізація знаходиться в *Додатку Б – ArchiveCompressorAdapter.java*.

Як можна побачити даний клас спадкується від абстрактного архіватора та при ініціалізації приймає в себе інший архіватор та компресор. Методи реалізовані поєднанням дій описаних раніше об'єктів: результат роботи компресора передається до архіватора й у зворотному напрямку. Проміжні файли зберігаються так само в директорії *temp* та після виконання видаляються. Даний тип архівів можна так само отримати через фабричний метод описаний в минулому підрозділі, вказавши тип компресору. Поведінка такого архіватора не буде відрізнятися від звичайного, вже знайомого користувачу.

Загалом даний підхід дозволив зменшити загальну кількість коду та використовувати описану раніше логіку. Переходимо до наступної частини.

4.2.4. Реалізація розпакування архіву

Суть розпакування полягає у вивільненні даних архіву за заданим шляхом, при чому бажано щоб такі дані як ієрархія, назви, мета-дані залишалися максимально наближеними до оригіналу. Функціонал

відповідальний за експорт файлів з архіву знаходиться в пакеті *archiver_api.output*.

Однією з вимог до системи роботи з архівами було використання шаблону відвідувача, тому було вирішено саме в цій частині його реалізувати. Логіка буде будуватися навколо описаного раніше класу *Entity* (знаходиться в тому ж самому пакеті), котрий використовується як DTO що містить в собі повну інформацію про певний об'єкт архіву. *Entity* може приймати в себе *Visitor*'а за допомоги методу *accept()*, де в якості параметру передається сам відвідувач. В середині цього методу викликається метод *visit()*, отриманого відвідувача та в якості параметру передається *Entity* в якому викликано даний метод (*Додаток Б – Visitor.java*).

Відвідувач має в собі лише один метод, котрий приймає в себе об'єкт класу *Entity*. В майбутньому можна додати типізацію до даного інтерфейсу, щоб можна було його використовувати з іншими класами. Переходимо до єдиної реалізації (*Додаток Б – ExportVisitor.java*).

При ініціалізації об'єкту конструктор класу приймає в себе шлях, куди буде розпаковано архів. Далі було створено метод *export()*, що приймає в себе список сутностей. В середині він починає по черзі викликати метод *accept()* у сутностей і, як параметр, передавати посилання на самого себе. Як було описано раніше в середині сутності викликається метод *visit()* та в якості параметру передається посилання на сутність.

В самому методі *visit()* на початку поєднується шлях експортування із внутрішнім шляхом функції. Після чого створюються директорії ієрархії (за необхідності). Якщо за отриманим шляхом вже існує файл, то викликається утилітарний метод *getFreePath()* класу *FileUtils*, що видозмінює посилання для можливості зберегти файл без конфліктів та перезапису вже існуючого.

Таким чином функція розпакування була реалізована як додаткова функція, заснована на базових функціях описаних в першому підрозділі та

служує гарним прикладом того як можна робити більш складну логіку на основі вже написаної без зміни попереднього коду.

На цьому частина програмно-прикладного інтерфейсу системи архівації файлів завершено. Далі буде описано інший компонент застосунку – файлообмінювач.

4.3. Файлообмінювач

Файлообмінювач представлений як клас, що у собі має два внутрішні класи: *FileSender* і *FileReceiver*, для надсилання та отримання повідомлення відповідно. Останній в собі має поле що відповідає назві директорії, куди будуть приходити отримані повідомлення, за не наявності даної – відбувається ініціалізація її. Полями основного класу є екземпляри об'єктів внутрішніх класів, а також присутня константа – порт який резервується для роботи сервісу. Реалізація класу знаходиться в *Додатку Б – P2PFileTransfer.java*.

Метод *init()* використовується для початку прослуховування порту, можливість встановлення з'єднання. Як вже було сказано в підготовчому етапі: під час цього етапу, потік виконання програми блокується, що унеможлиблює роботу в цей момент з іншими компонентами застосунку. Для того щоб вивести потік з цього стану, встановлюється змінна *soTimeout*, що вказує час через який закінчиться прослуховування порту та викинеться виключення *SocketTimeoutException*, за відсутності наявних підключень.

Окрім вище описаного методу, є ще один - *start()*. Він створює й повертає посилання на окремий потік, в середині якого запускає метод *init()*. Після проходження часу *soTimeout*, або обробки отриманого запиту, сервер знову переходить в стан очікування нових з'єднань. Система буде перебувати в цьому стані до тих пір, доки виконання потоку не буде перервано (реалізовано механізм переривання).

При отриманні запиту, викликається метод *receive()* відповідного екземпляру класу, де параметром передається ендпоінт з'єднання. Даний

метод робить ініціалізацію файлу з отриманими мета-даними від іншого клієнту (назва файлу, дата створення, дата останнього змінення), після чого вже відбувається запис інформації в сам файл.

Для того щоб передати повідомлення, треба викликати метод *send()*, параметрами якого треба вказати IP-адресу та шлях до файлу, що буде передаватися. Далі буде зроблена перевірка на коректність файлу, після чого запит буде передано до екземпляру класу *FileSender*. Його ціль: встановити з'єднання з серверною частиною іншого клієнту та передати файл у послідовності описаній у підготовчому етапі розробки. З основних моментів слід нагадати що файл передається розділений блоками по 4 кілобайти.

4.4. Реалізація користувацького інтерфейсу

Взаємодія з користувачем реалізована окремим компонентом, що знаходиться в пакеті *archiver_ui.console*. В основу покладено взаємодія через стандартні потоки вводу/виводу (*System.in* та *System.out*). Проте, незважаючи на те що даний спосіб є доволі простим в реалізації варіантом отримання інформації від клієнту, було розроблено декілька цікавих рішень для полегшення створення, додавання та видозміни вже існуючих блоків. Деталі них описано нижче.

4.4.1. Розбиття на середовища

Для того щоб зробити написання запитів більш простим та інтуїтивним, для користувача було прийнято рішення розбити застосунок на блоки – середовища (*Додаток Б – Environment.java*).

Ці компоненти: знаходяться в пакеті *archive_ui.console.environments*. Кожне середовище складається з потоку вводу, та потоку принтера, що виводить результат виконання до інтерфейсу. Також середовища мають у собі поле *inputType*, що вказує користувачу саме де він знаходиться. Метод *start()* виконує роль запуску середовища, в якому користувач перебуває до моменту отримання виходу – надсилання команди *exit*.

В кожному середовищі є власний набір команд, що працює тільки в заданому середовищі. Для кращого розуміння можна уявляти середовища як папки, кожна з яких містить власні файли. Як і шлях до папки містить корінь, так і середовища починаються з головного (*Додаток Б – MainEnvironment.java*). Вихід з головного середовища, означає вихід з програми.

Тут користувач може обрати з яким компонентом він хоче взаємодіяти. В даній версії програмного забезпечення передбачено три компоненти взаємодії:

- Помічник (helper/help)
- Архіватор (archive/archiver)
- Файлообмінювач (file transfer/fileTransfer/transfer)

В помічнику можна дізнатися які середовища існують, як їх викликати так які команди вони в собі мають. Для першого випадку треба в цьому середовищі ввести команду *help*, а для другого ввести назву середовища.

Архіватор та Файлообмінювач являють собою візуальне представлення функціоналу який, було описану в минулих підрозділах. Так архіватор має наступні команди: *new*, *add*, *remove*, *show*, *extract*. Дії того, що вони виконують зрозумілі з назви. В свою чергу, файлообмінювач має такий перелік дій: *send*, *init*. Остання переводить клієнт у стан приймання інформації від інших користувачів через першу команду.

Також виникає питання, а як саме інформація передається від клієнту до компонентів. Для цього було розроблено друге рішення – генерація форм.

4.4.2. Генерація форм

В основу цього компоненту лягли елементи, що знаходяться в пакеті *archiver_ui.console.input*. Загалом все розроблено на основі шаблону будівельник. Найменшою одиницею в даній системі є поле, куди користувач

записує дані. Для кращого розуміння нижче знаходиться приклад, що відповідаю візуальному відображенню поля:

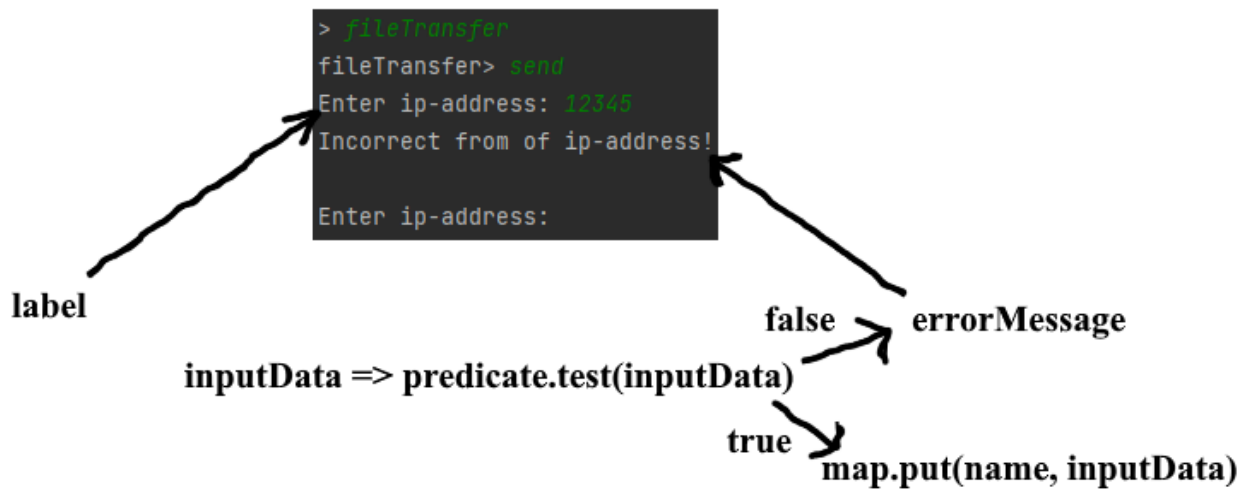


Рис. 4.3. Приклад поля

Кожне поле складається з етикетки (*label*), де вказується інформація перед очікуванням введення від користувача. Предикатора (*predicate*), що перевіряє введені дані на задані умови під час ініціалізації. Повідомлення про помилку (*errorMessage*), якщо умови предикатора не було виконано. Також кожне поле містить власне ім'я (*name*) для витягнення даних цих полів під час виконання повноцінних форм. Показано даний клас в Додатку Б – *InputField.java*.

Крім того, було додано можливість створення полів, що можуть приймати в себе декілька значень. За це відповідають два атрибути: *isMultiple* та *splitter*. Першим є значення булевого типу, що за замовчуванням встановлюється в *false*, для того щоб використовувати дану функцію треба перевести прапорець у стан *true*. Другим атрибутом є розділювач, що ділить між собою усі запити, встановленим при ініціалізації символом (або групою символів). Це зроблено через те, що тип повернутого значення має бути звичайний рядок.

Слід також зазначити, що під час ініціалізації треба вказати з потік вводу – звідки інформація буде зчитуватися та принтер – куди буде виводитися.

Для запуску поля треба використати метод *start()*. Дія поля закінчить своє виконання у той момент, коли отримує від користувача коректне значення. Таким чином поля можуть утворювати між собою послідовності отримання інформації, або інакше кажучи – форму (*Додаток Б – InputForm.java*).

Форми складаються зі списку полів та словника де парою ключ-значення виступає рядок-рядок. Під час ініціалізації ззовні приходить список полів та створюється пустий словник результатів. Для того щоб запустити форму викликається метод *execute()*, що повертає у відповідь мапу (словник) значень.

Сама мапа формується під час послідовного запуску форм й у разі отримання відповіді, записує її до результатів і як ключ вказує ім'я поля. Це дозволяє запустити форму, а потім з отриманих результатів зручно сформувати запит й відправити його до подальших сервісів.

Однак, можна побачити, що створення форм в ручному вигляді є не дуже зручним. Це відбувається через монотонне створення полів, кожне з яких має в собі велику кількість параметрів при створенні. В додаток, такі списки буде дуже важко видозмінювати, в разі потреби. Тому було вирішено й цей момент зробити простіше – створення будівника форм (*Додаток Б – FormBuilder.java*).

Даний клас побудований за шаблоном будівника. Так має в собі наступні значення: потік вводу та принтер – для використання їх під час створення форми, та списку полів – що потім буде передаватися для створення самої форми. Якщо спростити, то білдер складається з таких функцій: встановити потік вводу та принтер (за замовчуванням вони приймають в себе значення стандартних потоків вводу/виводу системи) і додаванням полів. Останнє присутнє в трьох варіантах: напряму додати поле, створити й додати поле що приймає в себе одне значення, створити й додати поле що приймає в себе декілька значень. Після того як список був остаточно сформований, треба здійснити виклик методу *build()*, що поверне готову для використання форму.

Таким чином було створено фундамент, що пришвидшить розробку користувацького інтерфейсу в разі та дасть можливість його легко модифікувати та видозмінювати. Далі показано приклад створення форми з використанням усіх описаних вище в цьому розділі компонентів:

```
Map<String, String> res = new FormBuilder(in, out)
    .addField("Enter ip-address: ", "ip", InputPredicates.isCorrectIp, "Incorrect form of ip-
address!")
    .addField("Enter file path: ", "filePath", InputPredicates.isExistedFile, "Path is
incorrect or file does not exist!")
    .build()
    .execute();
```

Рис. 4.4. Приклад створення форми

4.4.3. Локалізація користувацького інтерфейсу

Задля привернення уваги користувачів з різних країн, було вирішено додати функціонал локалізації застосунку. В залежності від регіону де було здійснено запуск, буде вибиратися конкретний ресурс та на основі нього буде формуватися вивід користувачу. Реалізовано це на функціоналі влаштованих бібліотек java, а саме на *ResourceBundle* та *Locale*.

Для того щоб використовувати перший клас, треба під час створення вказати шлях до «сімейства» ресурсів та (опціонально) локалізацію (об'єкт класу *Locale*). Сама інформація зберігається у файлах типу *properties* (Рис. 4.5.) й дістається методом *getString()*.

```
var1 = Single line!
var2 = \
    Single line\
    too!
var3 = Multi line 1!\n\
    Multi line 2!\n\ Multi line 3!
```

Рис. 4.5. Приклад файлу типу *properties*

Так як *ResourceBundle* є *immutable*, то для полегшення реалізації зміни мови під час роботи системи, було розроблено клас за шаблоном *Singleton*

через який буде відбуватися вся взаємодія з мовними файлами властивостей. Клас має всередині константу шляху до місця збереження локалізації та під час створення встановлюється локалізація за замовчуванням системи. Для зміни локалізації присутній метод *changeResource()*, де в якості параметру приходить нова локалізація, після чого відбувається перезапис змінної *currentBundle*. Екземпляр цього класу існує лише в одному екземплярі задля унеможливлення виникнення ситуації коли кожен компонент програми використовує власну локалізацію. Повний код цього класу знаходиться в *Додатку Б – ResourceManager.java*.

На момент першої версії існує дві локалізації: англійська (за замовчування) та українська. В майбутньому кількість буде збільшуватися, а сформований фундамент тільки буде сприяти цьому, бо додавання нових мовних пакетів можливе не буде великою проблемою й може виконуватися людиною, що не є спеціалістом у сфері програмування.

Опис розробки на цьому завершено. Для більш детального ознайомлення в *Додатку Б – Репозиторій* вказано посилання на github-репозиторій.

ВИСНОВОК

В результаті роботи було отримано робочий прототип що може йти у відкритий доступ для подальшого використання. Застосунок є абсолютно справним, має відкритий код та зрозумілу архітектуру, що дає змогу майбутній спільноті брати безпосередню участь розробці та підтримці системи.

Під час розробки було продемонстровано усі етапи проектування: від огляду існуючих рішень, до розробки додаткового функціоналу, такого як система логування дій файлообмінювача та використання мовних пакетів для швидкого додавання нових локалізацій у застосунок.

На етапі проектування було зображено наступні діаграми: діаграма розгортання, діаграма прецедентів та діаграма діяльності, діаграма класів та діаграма послідовності, а також діаграма станів. Отримані моделі сформували чітку уяву того, як має виглядати система, що дало змогу вирішити майбутні проблеми, ще на підготовчому етапі. Визначившись архітектурою застосунку та які при цьому мови програмування (й бібліотеки відповідно) будуть використовуватися, почалася безпосередньо розробка.

На етапі розробки було написано систему на мові Java з використанням додаткових бібліотек: *Commons Compress* та *Log4j2*. Отримана система має в собі наступні шаблони проектування: стратегія, адаптер, фабричний метод, фасад та відвідувач. Описані вище рішення повністю задовільнили поставлене перед виконанням технічне завдання.

Таким чином застосунок може використовуватися як в домашніх умовах, так і в робочих командах й проектах. Також, невід’ємною перевагою є використання архітектури peer-to-peer, що є ідеальним рішенням для прямого обміну файлами між користувачами по мережі.

Через дотримання принципів об’єктно-орієнтовного програмування, принципів SOLID та використання шаблонів програмування – отримана система є легкою для входження нових розробників, має в собі можливість без

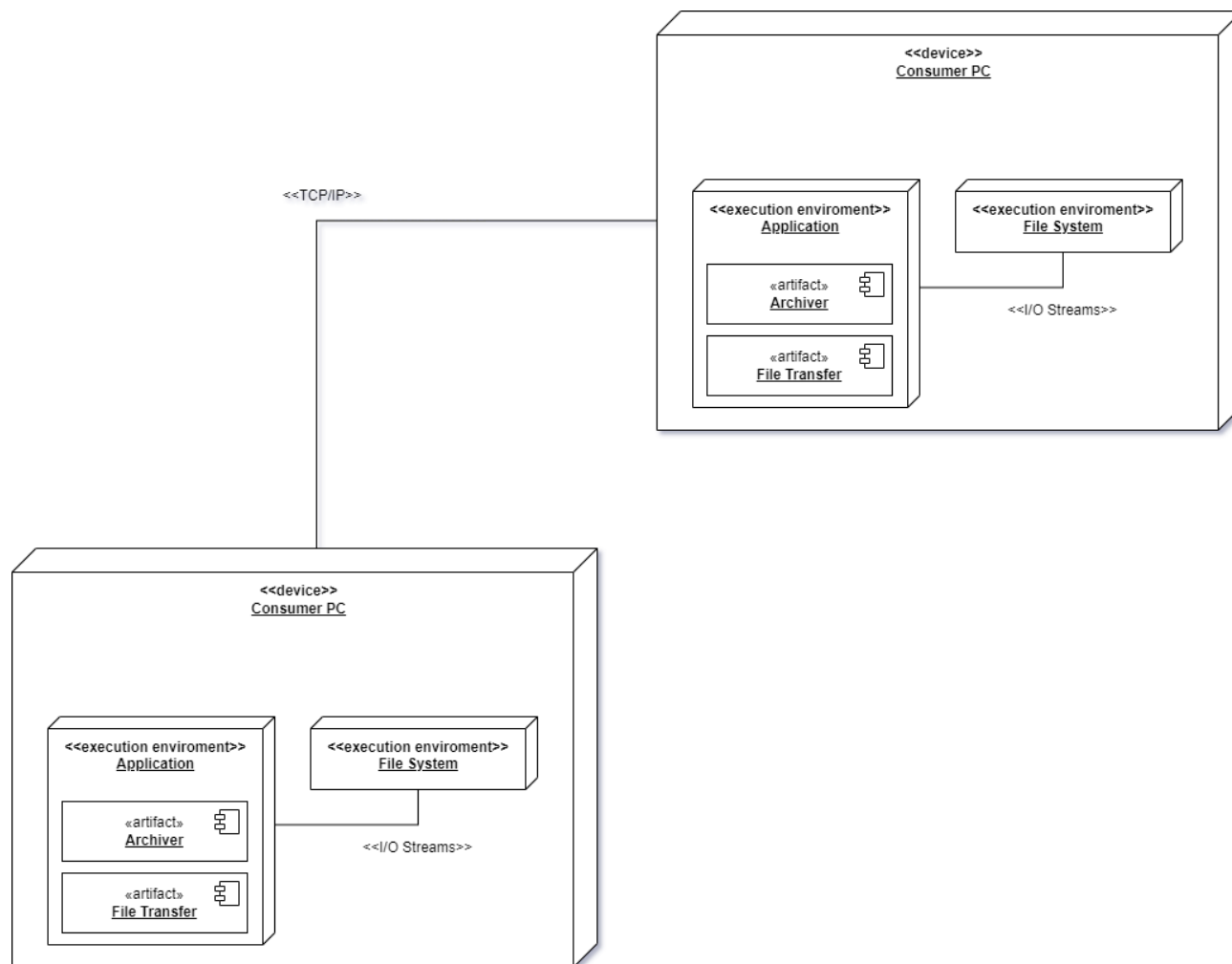
ускладнень нарощувати новий функціонал та удосконалювати старий й замінювати одні модулі програми на інші за мінімальні проміжки часу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

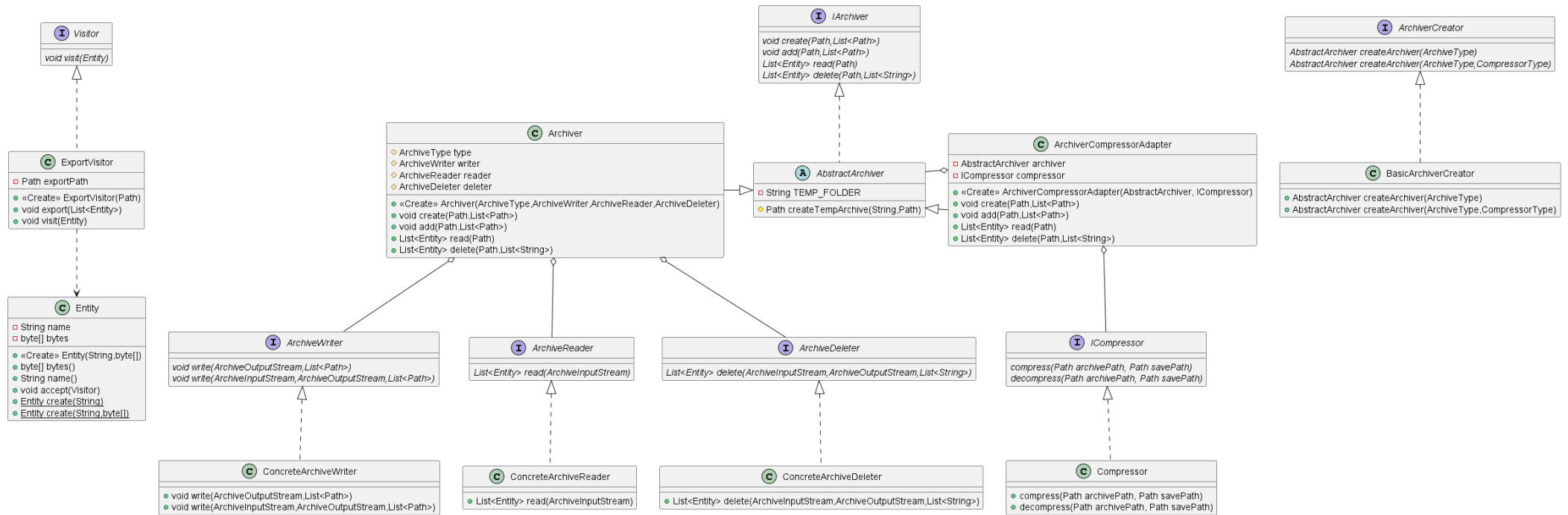
- [1] Ethan, “The Best 7 File Archivers for Windows – Boost Your File Compression Game!”, SYSGEEKER, 2023. [Online]. Available: [The Best 7 Filers for Windows - Boost Your File Compression Game! \(sysgeeker.com\)](https://sysgeeker.com/the-best-7-file-archivers-for-windows-boost-your-file-compression-game/). Accessed on: August 9, 2023.
- [2] Boldson, “Windows 11, built-in archiver, File Explorer, Neowin, WinRAR, NanaZIP, 23H2 update, data compression, 7-zip, rar”, GameGPU, 2023. [Online]. Available: [Windows 11, built-in archiver, File Explorer, Neowin, WinRAR, NanaZIP, 23H2 update, data compression, 7-zip, rar. | Hardware | NEWS \(gamegpu.tech\)](https://gamegpu.tech/windows-11-built-in-archiver-file-explorer-neowin-winrar-nanazip-23h2-update-data-compression-7-zip-rar/). Accessed on: August 27, 2023.
- [3] Sofia Elizabella Wyciślik-Wilson, “Best file compression software in 2023”, techradar, 2023. [Online]. Available: [Best file compression software in 2023 | TechRadar](https://www.techradar.com/best/file-compression-software-in-2023).
- [4] О. Швець, *Занурення в Патерни Проектування*, Refactoring.Guru, 2018.
- [5] Richard M Reese, *Learning Network Programming with Java*, Packt Publishing, 2015.
- [6] Lucidchart Blog, “Types of UML Diagrams”, Lucidchart, 2023. [Online]. Available: [Introducing Types of UML Diagrams | Lucidchart Blog](https://lucidchart.com/blog/types-of-uml-diagrams/)
- [7] Creately, “UML Diagram Types Guide: Learn About All Types of UML Diagrams with Examples”, creately, 2022. [Online]. Available: [UML Diagram Types | Learn About All 14 Types of UML Diagrams \(creately.com\)](https://creately.com/blog/uml-diagram-types/)
- [8] Apache Commons, *Apache Commons Compress*, 2023. [Online]. Available: [Commons Compress – Commons Compress User Guide \(apache.org\)](https://commons.apache.org/compress/). Accessed on: November 17, 2023.
- [9] К. Хорстманн, *Java. Библиотека профессионала, том 2. Расширенные средства программирования*, 11-е изд. СПб: ООО "Диалектика", 2020.
- [10] HeptaDecane, “File Transfer via Java Sockets”, Medium, 2020. [Online]. Available: [File Transfer via Java Sockets. I suppose you have been doing some... | by HeptaDecane | Medium](https://medium.com/@heptadecane/file-transfer-via-java-sockets-i-suppose-you-have-been-doing-some...-by-heptadecane)

ДОДАТКИ

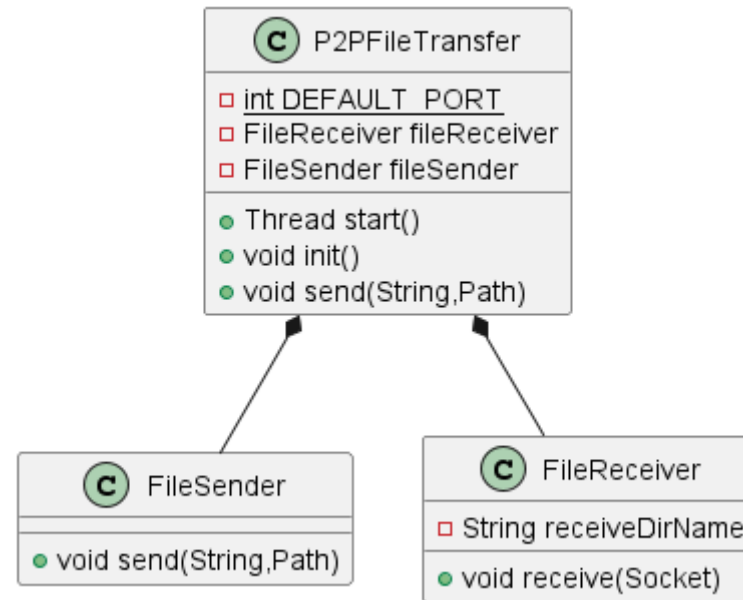
Додаток А



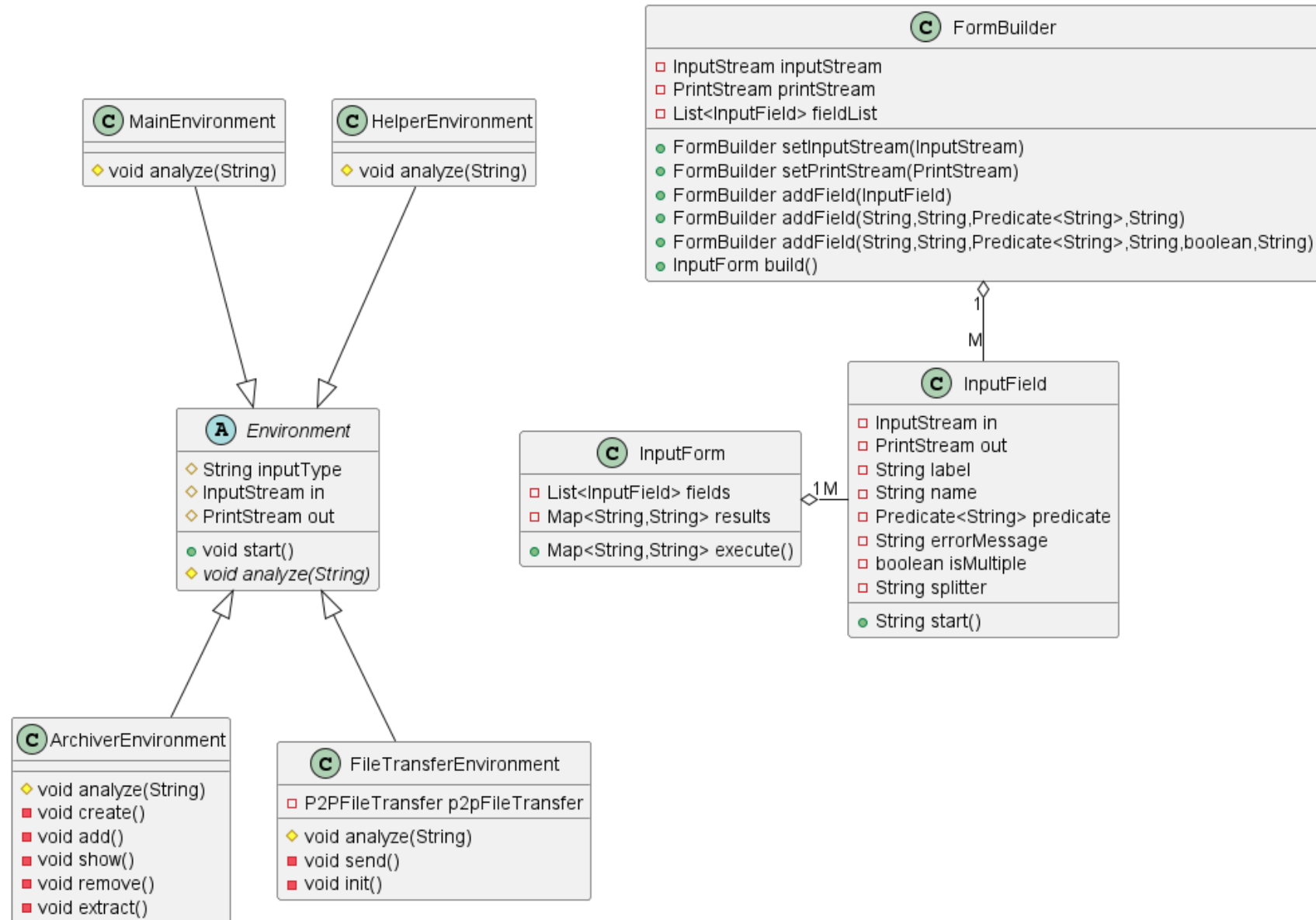
Deployment diagram



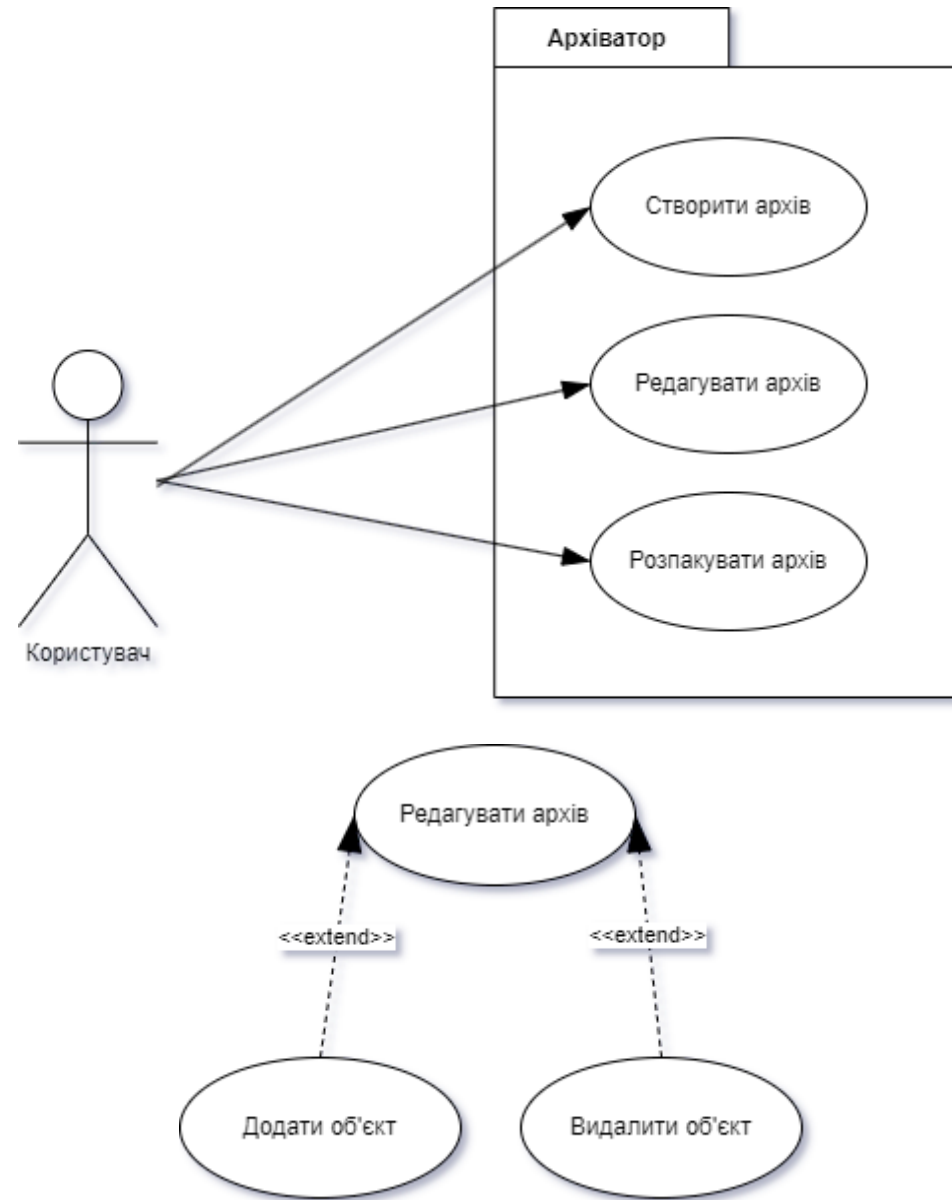
Class diagram (Archiver)



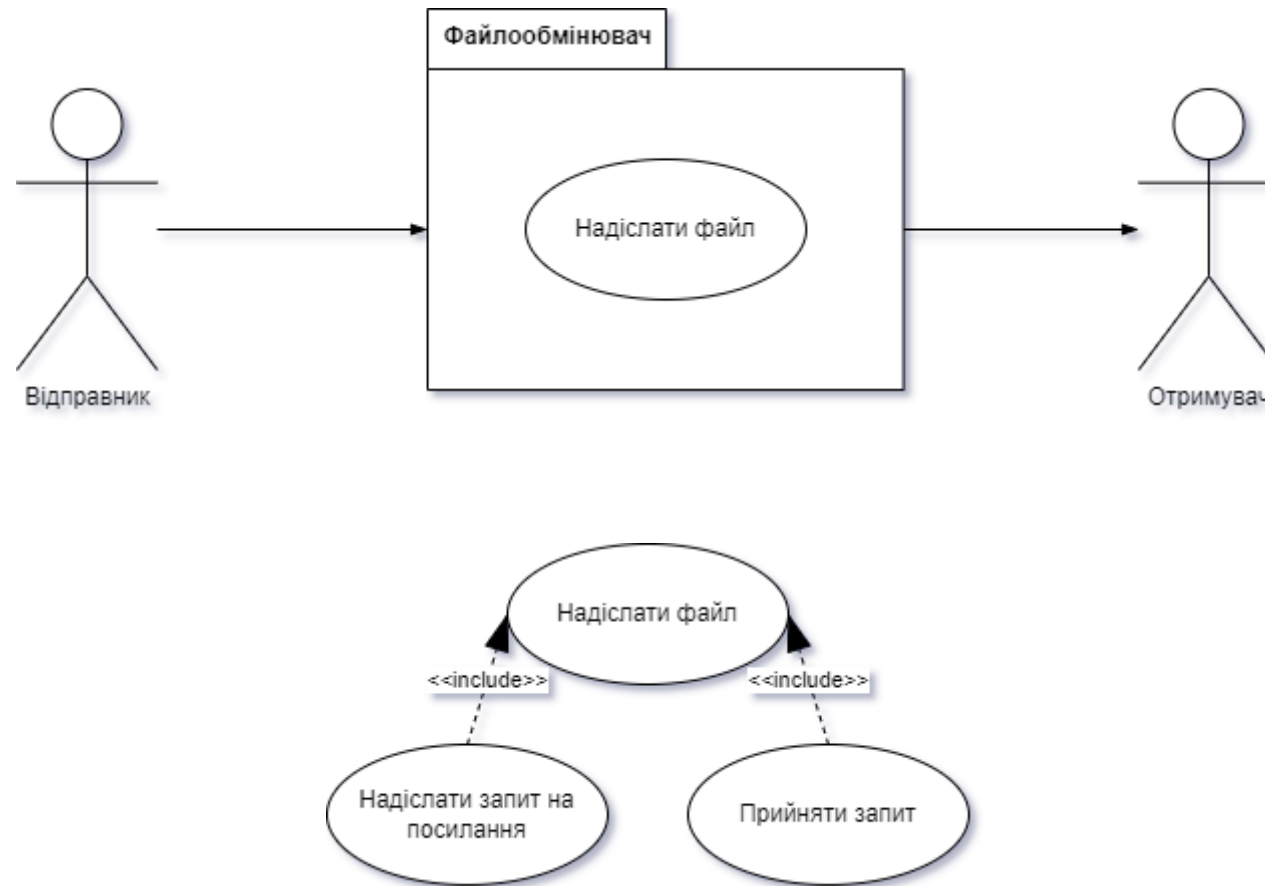
Class diagram (File Transfer)



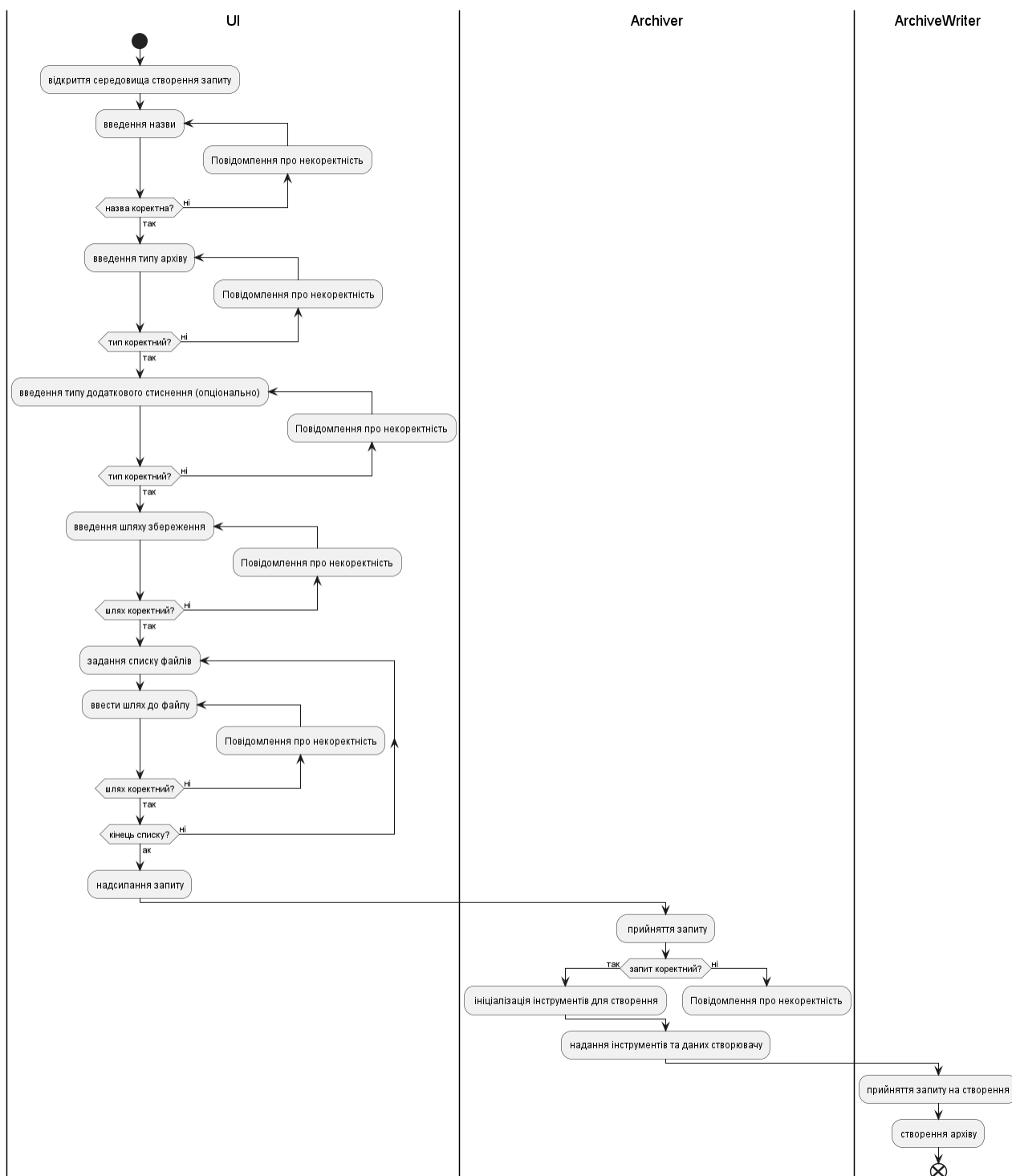
Class diagram (Console UI)



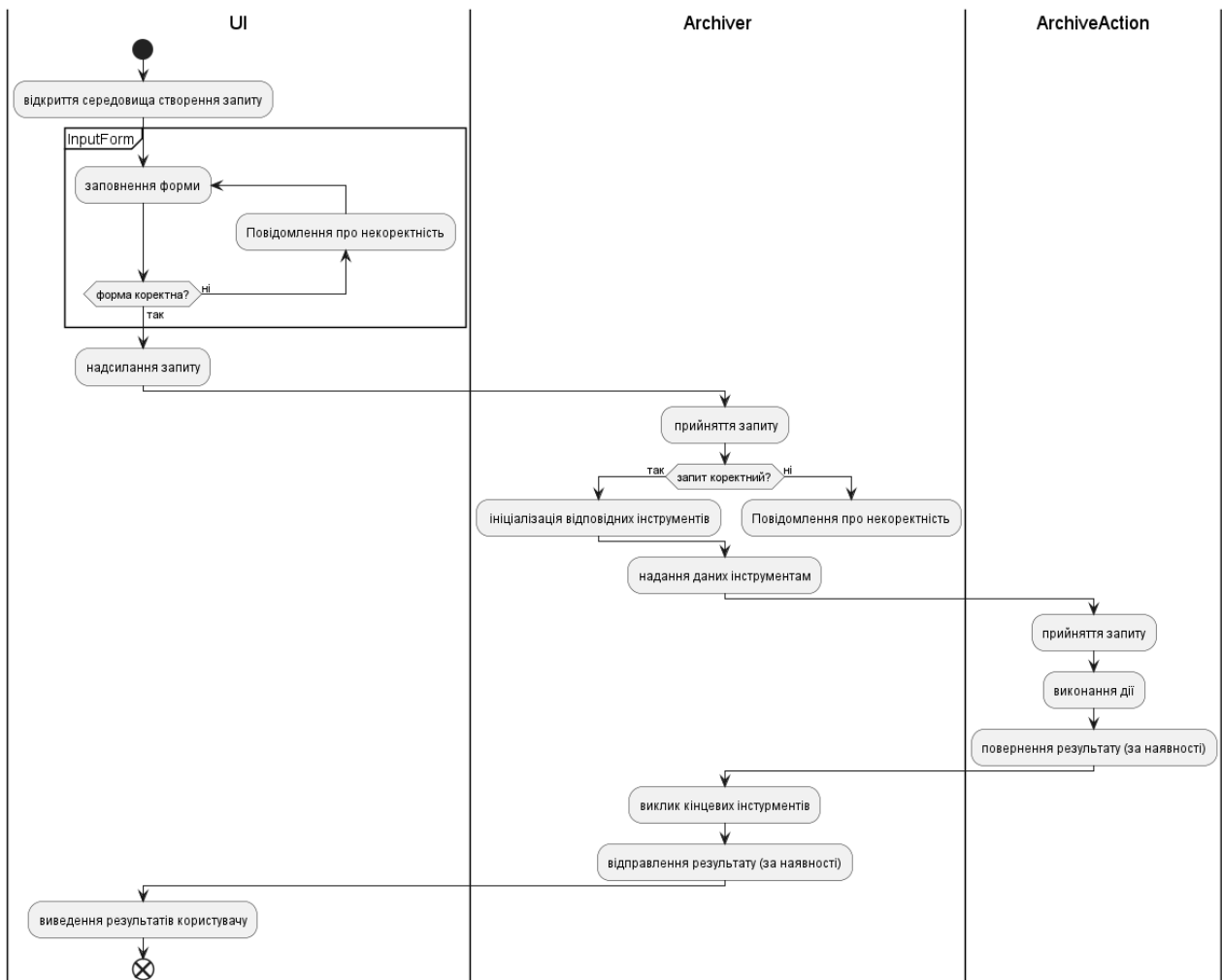
Use-case diagram (Archiver)



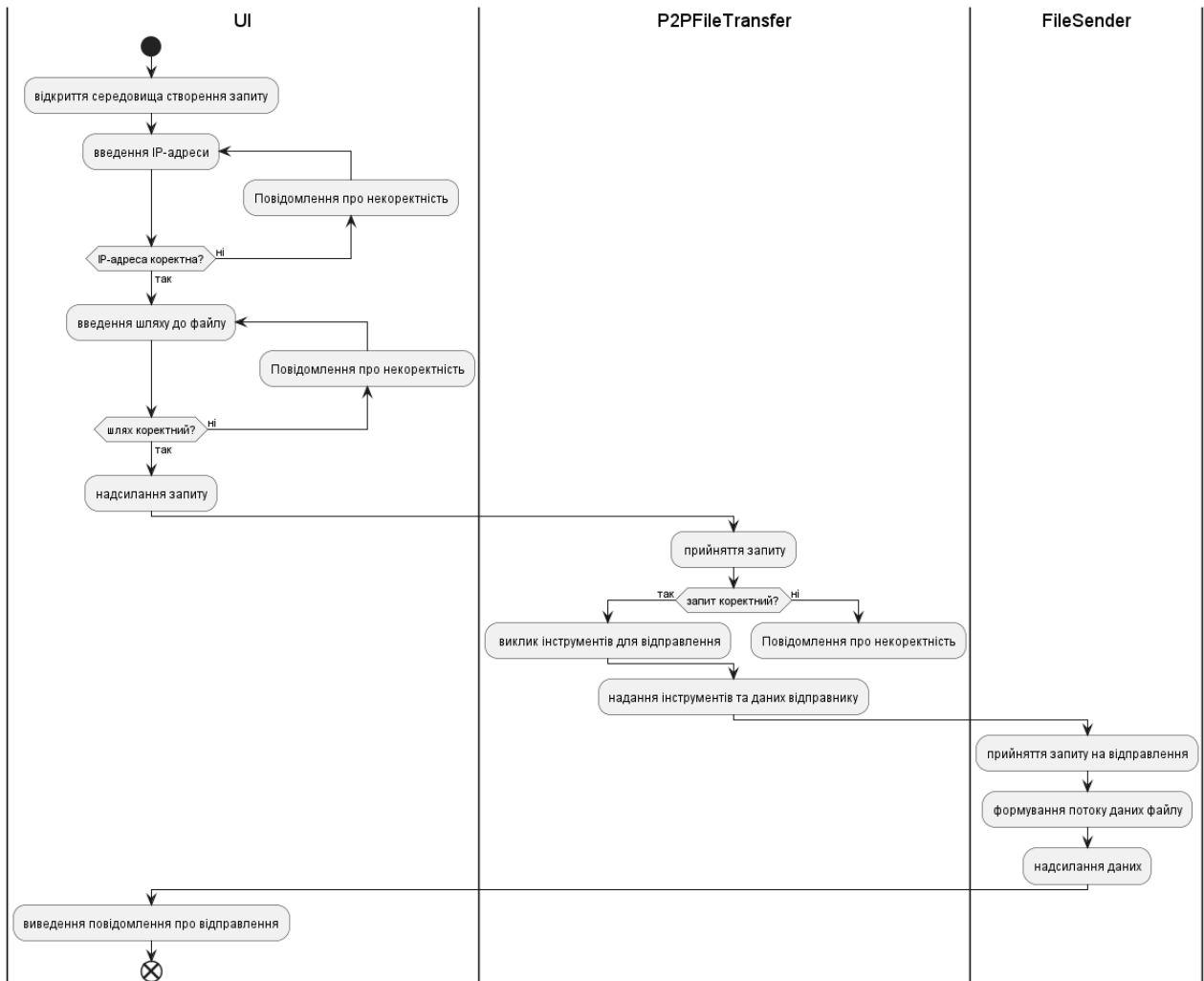
Use-case diagram (File Transfer)



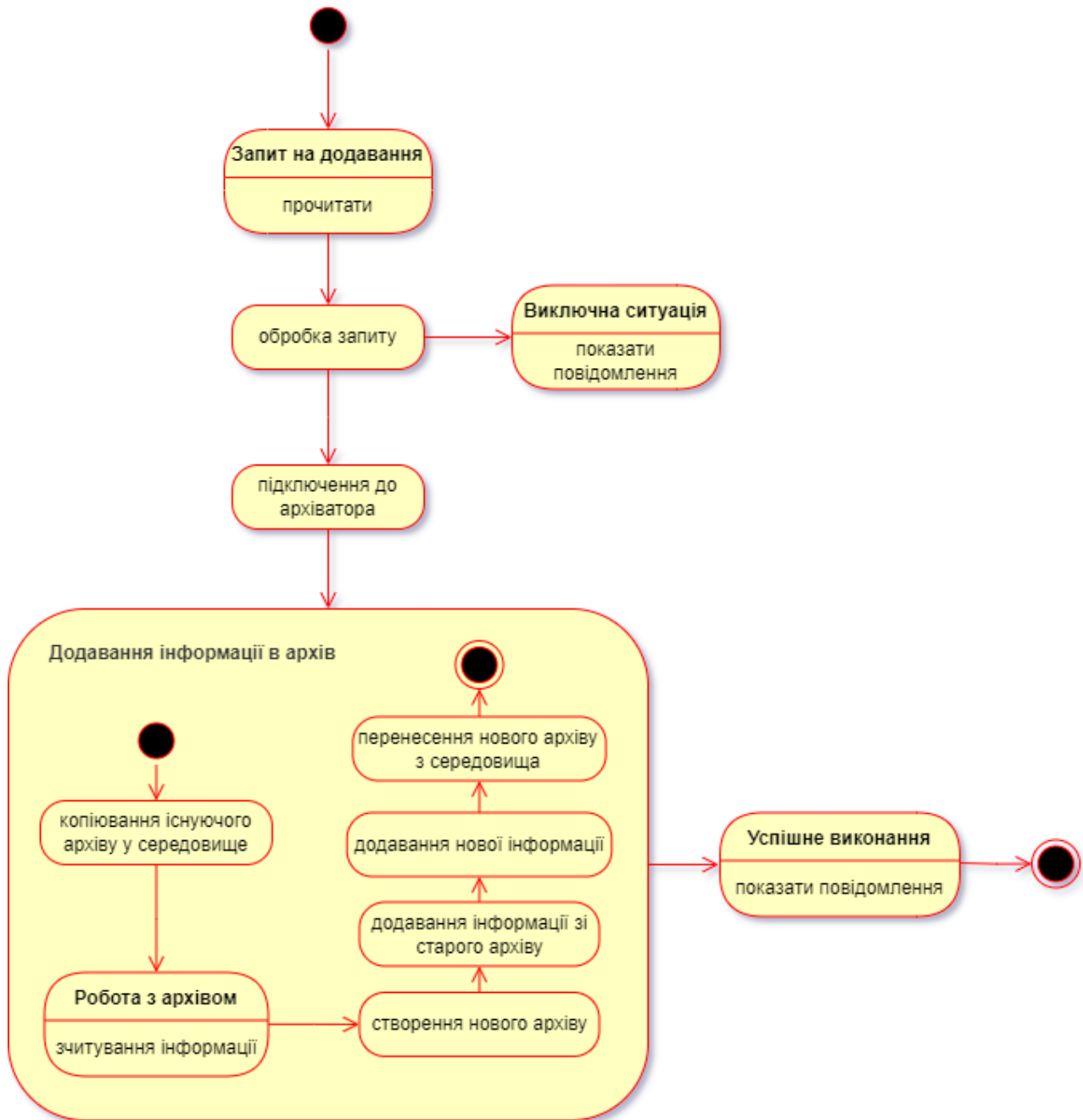
Activity diagram (Archive Creation)



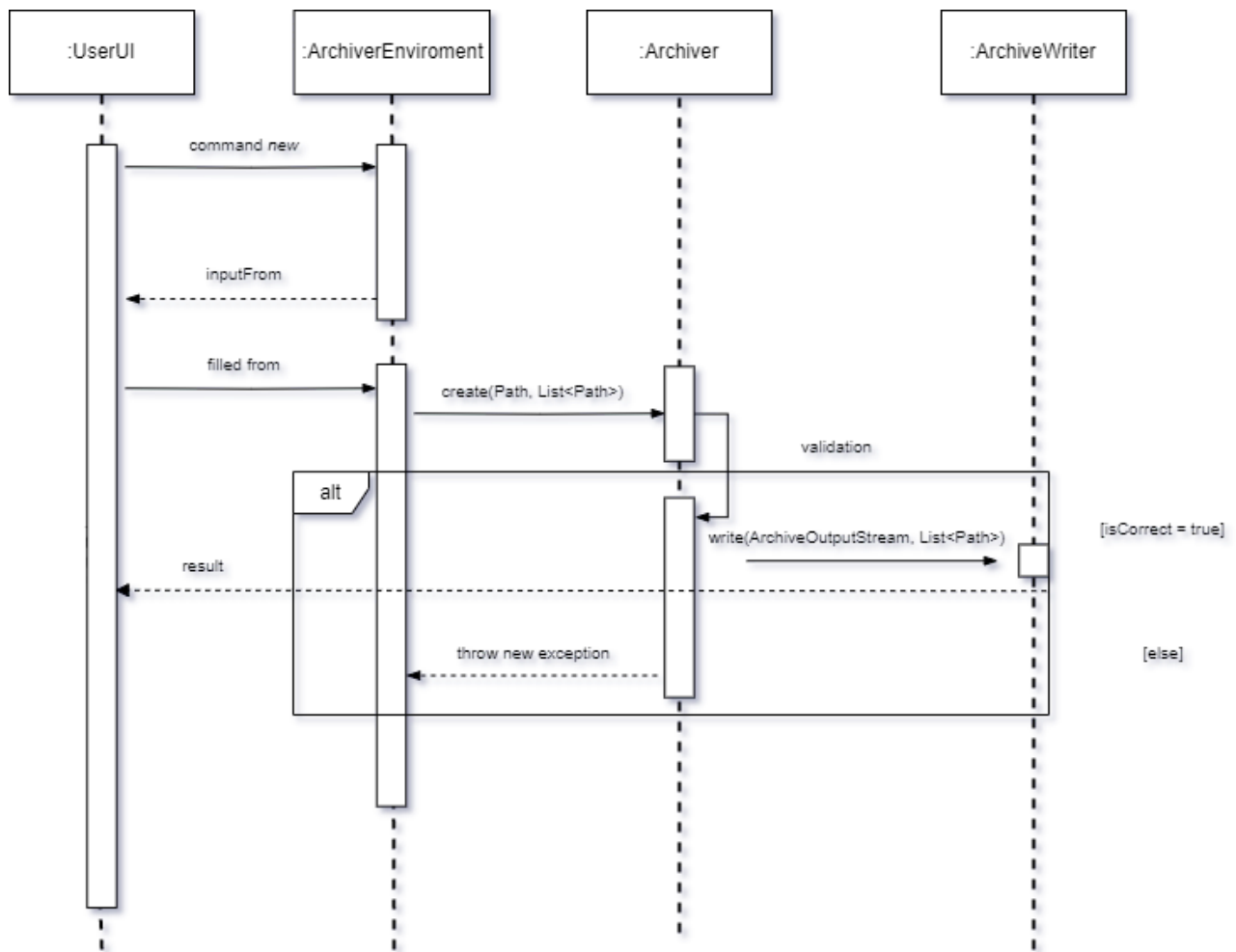
Activity diagram (Archive Basic)



Activity diagram (File Transfer – send)



State diagram (Archive Adding)



Sequence diagram (Archive Creation)

Додаток Б

```
public interface ArchiveWriter {  
    void write(ArchiveOutputStream aos, List<Path> pathList) throws IOException;  
    void write(ArchiveInputStream ais, ArchiveOutputStream aos, List<Path> pathList) throws IOException;  
}
```

ArchiveWriter.java

```
public interface ArchiveReader {  
    List<Entity> readBasic(ArchiveInputStream ais) throws IOException;  
    List<Entity> readFull(ArchiveInputStream ais) throws IOException;  
}
```

ArchiveReader.java

```
public interface ArchiveDeleter {  
    List<Entity> delete(ArchiveInputStream ais, ArchiveOutputStream aos, List<String> entityNames) throws  
IOException;  
}
```

ArchiveDeleter.java

```

public class BasicArchiveWriter implements ArchiveWriter {
    @Override
    public void write(ArchiveOutputStream aos, List<Path> pathList) throws IOException {
        for (Path path : pathList) {
            if (Files.isDirectory(path)) {
                List<Path> localPathList = getAllFilePathsFromDirectory(path);
                for (Path localPath: localPathList) {
                    String cutName = cutPathName(path, localPath);
                    ArchiveEntry newEntry = aos.createArchiveEntry(localPath, cutName);

                    aos.putArchiveEntry(newEntry);
                    try (InputStream is = Files.newInputStream(localPath)) {
                        IOUtils.copy(is, aos);
                    }
                    aos.closeArchiveEntry();
                }
            } else {
                ArchiveEntry newEntry = aos.createArchiveEntry(path, path.getFileName().toString());
                aos.putArchiveEntry(newEntry);
                try (InputStream is = Files.newInputStream(path)) {
                    IOUtils.copy(is, aos);
                }
                aos.closeArchiveEntry();
            }
        }
    }

    @Override
    public void write(ArchiveInputStream ais, ArchiveOutputStream aos, List<Path> pathList) throws
IOException {
        ArchiveEntry currentEntry;
        List<String> namesList = new LinkedList<>();
        for (Path path : pathList) {
            if (Files.isDirectory(path)) {
                List<Path> localPaths = getAllFilePathsFromDirectory(path);
                namesList.addAll(
                    localPaths.stream().map(localPath -> cutPathName(path, localPath)).toList()
                );
            } else {
                namesList.add(path.getFileName().toString());
            }
        }
        while ((currentEntry = ais.getNextEntry()) != null) {
            String entryName = Path.of(currentEntry.getName()).toString();

            if (!namesList.contains(entryName)) {
                aos.putArchiveEntry(currentEntry);
                IOUtils.copy(ais, aos);
                aos.closeArchiveEntry();
            }
        }
        write(aos, pathList);
    }

    private String cutPathName(Path rootPath, Path filePath) {
        return filePath.toString()
            .replace(rootPath.getParent().toString(), "");
    }

    private List<Path> getAllFilePathsFromDirectory(Path dir) throws IOException {
        try (Stream<Path> walk = Files.walk(dir)) {
            return walk.filter(Files::isRegularFile).toList();
        }
    }
}

```

BasicArchiveWriter.java

```

public class BasicArchiveReader implements ArchiveReader {
    @Override
    public List<Entity> readBasic(ArchiveInputStream ais) throws IOException {
        List<Entity> entityList = new LinkedList<>();

        ArchiveEntry currentEntry;
        while ((currentEntry = ais.getNextEntry()) != null) {
            entityList.add(Entity.create(currentEntry.getName()));
        }

        return entityList;
    }

    @Override
    public List<Entity> readFull(ArchiveInputStream ais) throws IOException {
        List<Entity> entityList = new LinkedList<>();

        ArchiveEntry currentEntry;
        while ((currentEntry = ais.getNextEntry()) != null) {
            entityList.add(Entity.create(currentEntry.getName(), currentEntry.getLastModifiedDate(),
IOUtils.toByteArray(ais)));
        }

        return entityList;
    }
}

```

BasicArchiveReader.java

```

public class BasicArchiveDeleter implements ArchiveDeleter {
    @Override
    public List<Entity> delete(ArchiveInputStream ais, ArchiveOutputStream aos, List<String> entityNames)
throws IOException {
        List<Entity> deletedEntities = new LinkedList<>();

        ArchiveEntry currentEntry;
        while ((currentEntry = ais.getNextEntry()) != null) {
            if (entityNames.contains(currentEntry.getName())) {
                deletedEntities.add(Entity.create(currentEntry.getName()));
            } else {
                aos.putArchiveEntry(currentEntry);
                IOUtils.copy(ais, aos);
                aos.closeArchiveEntry();
            }
        }

        return deletedEntities;
    }
}

```

BasicArchiveDeleter.java


```

public record Entity(String name, Date lastModifiedDate, byte[] bytes) {
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Entity entity = (Entity) o;
        return Objects.equals(name, entity.name) && Arrays.equals(bytes, entity.bytes);
    }

    @Override
    public int hashCode() {
        int result = Objects.hash(name);
        result = 31 * result + Arrays.hashCode(bytes);
        return result;
    }

    @Override
    public String toString() {
        return "archiver_api.output.Entity{" +
            "name='" + name + '\'' +
            ", bytes=" + Arrays.toString(bytes) +
            '}';
    }

    public static Entity create(String name, Date lastModifiedDate, byte[] bytes) {
        return new Entity(name, lastModifiedDate, bytes);
    }

    public static Entity create(String name) {
        return new Entity(name, null, null);
    }

    public void accept(Visitor v) {
        v.visit(this);
    }
}

```

Entity.java

```
public interface IArchiver {
    void create(Path archivePath, List<Path> filePaths) throws IOException, ArchiveException;
    void add(Path archivePath, List<Path> filePaths) throws IOException, ArchiveException;
    List<Entity> read(Path archivePath, ArchiveReadingType archiveReadingType) throws IOException,
ArchiveException;
    List<Entity> delete(Path archivePath, List<String> fileNames) throws IOException, ArchiveException;
}
```

IArchiver.java

```
public abstract class AbstractArchiver implements IArchiver {
    private final String TEMP_FOLDER = "temp";

    private void initTempFolder() throws IOException{
        Path tempPath = Path.of(TEMP_FOLDER);
        if (Files.notExists(tempPath))
            Files.createDirectory(tempPath);
    }

    protected Path createTempArchive(String folder, Path archivePath) throws IOException {
        initTempFolder();
        Path folderPath = Path.of(TEMP_FOLDER, folder);
        if (Files.notExists(folderPath))
            Files.createDirectory(folderPath);

        Path tempArchivePath = folderPath.resolve(archivePath.getFileName().toString());
        tempArchivePath = FileUtils.getFreePath(tempArchivePath);

        return Files.createFile(tempArchivePath);
    }
}
```

AbstractArchiver.java

```
public interface ArchiverCreator {
    AbstractArchiver createArchiver(ArchiveType archiveType);
    AbstractArchiver createArchiver(ArchiveType archiveType, CompressorType compressorType);
}
```

ArchiveCreator.java

```
public class BasicArchiverCreator implements ArchiverCreator {
    @Override
    public AbstractArchiver createArchiver(ArchiveType archiveType) {
        return new Archiver(archiveType, new BasicArchiveWriter(), new BasicArchiveReader(), new
BasicArchiveDeleter());
    }

    @Override
    public AbstractArchiver createArchiver(ArchiveType archiveType, CompressorType compressorType) {
        return new ArchiverCompressorAdapter(
            createArchiver(archiveType),
            new Compressor(compressorType)
        );
    }
}
```

BasicArchiveCreator.java

```

public class Archiver extends AbstractArchiver {
    protected final ArchiveType archiveType;
    protected final ArchiveWriter writer;
    protected final ArchiveReader reader;
    protected final ArchiveDeleter deleter;

    public Archiver(ArchiveType archiveType, ArchiveWriter writer, ArchiveReader reader, ArchiveDeleter
deleter) {
        this.writer = writer;
        this.reader = reader;
        this.deleter = deleter;
        this.archiveType = archiveType;
    }

    @Override
    public void create(Path archivePath, List<Path> filePaths) throws IOException, ArchiveException {
        Path savedPath = FileUtils.getFreePath(archivePath);
        try {
            BufferedOutputStream bos = new BufferedOutputStream(Files.newOutputStream(savedPath));
            ArchiveOutputStream aos = new ArchiveStreamFactory()
                .createArchiveOutputStream(archiveType.name(), bos)
        } {
            writer.write(aos, filePaths);
        }
    }

    @Override
    public void add(Path archivePath, List<Path> filePaths) throws IOException, ArchiveException {
        Path tempArchivePath = createTempArchive("archives", archivePath);

        boolean isAdded = false;
        try {
            BufferedInputStream bis = new BufferedInputStream(Files.newInputStream(archivePath));
            BufferedOutputStream bos = new
BufferedOutputStream(Files.newOutputStream(tempArchivePath));
            ArchiveInputStream ais = new ArchiveStreamFactory()
                .createArchiveInputStream(archiveType.name(), bis);
            ArchiveOutputStream aos = new ArchiveStreamFactory()
                .createArchiveOutputStream(archiveType.name(), bos)

        } {
            writer.write(ais, aos, filePaths);
            isAdded = true;
        } finally {
            if (isAdded) {
                Files.copy(tempArchivePath, archivePath, StandardCopyOption.REPLACE_EXISTING);
            }
            Files.delete(tempArchivePath);
        }
    }

    @Override
    public List<Entity> read(Path archivePath, ArchiveReadingType archiverReadingType) throws IOException,
ArchiveException {
        try {
            BufferedInputStream bis = new BufferedInputStream(Files.newInputStream(archivePath));
            ArchiveInputStream ais = new ArchiveStreamFactory().createArchiveInputStream(bis)

        } {
            return switch (archiverReadingType) {
                case BASIC -> reader.readBasic(ais);
                case FULL -> reader.readFull(ais);
            };
        }
    }

    @Override
    public List<Entity> delete(Path archivePath, List<String> fileNames) throws IOException,
ArchiveException {
        Path tempArchivePath = createTempArchive("archives", archivePath);

```

```

        List<Entity> deletedEntities = null;
        try (
            BufferedInputStream bis = new BufferedInputStream(Files.newInputStream(archivePath));
            BufferedOutputStream bos = new
BufferedOutputStream(Files.newOutputStream(tempArchivePath));
            ArchiveInputStream ais = new ArchiveStreamFactory()
                .createArchiveInputStream(archiveType.name(), bis);
            ArchiveOutputStream aos = new ArchiveStreamFactory()
                .createArchiveOutputStream(archiveType.name(), bos)
        ) {
            deletedEntities = deleter.delete(ais, aos, fileNames);
            return deletedEntities;
        } finally {
            if (deletedEntities != null) {
                Files.copy(tempArchivePath, archivePath, StandardCopyOption.REPLACE_EXISTING);
            }
            Files.delete(tempArchivePath);
        }
    }
}

```

Archiver.java

```
public interface ICompressor {
    void compress(Path archivePath, Path savePath) throws IOException, CompressorException;
    void decompress(Path archivePath, Path savePath) throws IOException, CompressorException;
}
```

ICompressor.java

```
public class Compressor implements ICompressor {
    private final CompressorType type;

    public Compressor(CompressorType type) {
        this.type = type;
    }

    @Override
    public void compress(Path archivePath, Path savePath) throws IOException, CompressorException {
        try {
            BufferedInputStream bis = new BufferedInputStream(Files.newInputStream(archivePath));
            BufferedOutputStream bos = new BufferedOutputStream(Files.newOutputStream(savePath));
            CompressorOutputStream cos = new
CompressorStreamFactory().createCompressorOutputStream(type.name(), bos)
        ) {
            IOUtils.copy(bis, cos);
        }
    }

    @Override
    public void decompress(Path archivePath, Path savePath) throws IOException, CompressorException {
        try {
            BufferedOutputStream bos = new BufferedOutputStream(Files.newOutputStream(savePath));
            BufferedInputStream bis = new BufferedInputStream(Files.newInputStream(archivePath));
            CompressorInputStream cis = new
CompressorStreamFactory().createCompressorInputStream(type.name(), bis)
        ) {
            IOUtils.copy(cis, bos);
        }
    }
}
```

Compressor.java

```

public class ArchiverCompressorAdapter extends AbstractArchiver {
    private final AbstractArchiver archiver;
    private final Compressor compressor;
    public ArchiverCompressorAdapter(AbstractArchiver archiver, Compressor compressor) {
        this.archiver = archiver;
        this.compressor = compressor;
    }

    @Override
    public void create(Path archivePath, List<Path> filePaths) throws IOException, ArchiveException {
        Path tempPath = createTempArchive("compress", archivePath);
        archiver.create(tempPath, filePaths);
        try {
            compressor.compress(tempPath, archivePath);
        } catch (CompressorException e) {
            throw new RuntimeException(e);
        } finally {
            Files.delete(tempPath);
        }
    }

    @Override
    public void add(Path archivePath, List<Path> filePaths) throws IOException, ArchiveException {
        Path tempPath = createTempArchive("compress", archivePath);
        try {
            compressor.decompress(archivePath, tempPath);
            archiver.add(tempPath, filePaths);
            compressor.compress(tempPath, archivePath);
        } catch (CompressorException e) {
            throw new RuntimeException(e);
        } finally {
            Files.delete(tempPath);
        }
    }

    @Override
    public List<Entity> read(Path archivePath, ArchiveReadingType archiveReadingType) throws IOException,
ArchiveException {
        Path tempPath = createTempArchive("compress", archivePath);
        try {
            compressor.decompress(archivePath, tempPath);
            return archiver.read(tempPath, archiveReadingType);
        } catch (CompressorException e) {
            throw new RuntimeException(e);
        } finally {
            Files.delete(tempPath);
        }
    }

    @Override
    public List<Entity> delete(Path archivePath, List<String> fileNames) throws IOException,
ArchiveException {
        Path tempPath = createTempArchive("compress", archivePath);
        List<Entity> deletedEntities;
        try {
            compressor.decompress(archivePath, tempPath);
            deletedEntities = archiver.delete(tempPath, fileNames);
            compressor.compress(tempPath, archivePath);

            return deletedEntities;
        } catch (CompressorException e) {
            throw new RuntimeException(e);
        } finally {
            Files.delete(tempPath);
        }
    }
}

```

```
public interface Visitor {
    void visit(Entity entity);
}
```

Visitor.java

```
public class ExportVisitor implements Visitor {
    private final Path exportPath;

    public ExportVisitor(Path exportPath) {
        if (!Files.isDirectory(exportPath)) {
            throw new IllegalArgumentException("Path must be a directory!");
        }
        this.exportPath = exportPath;
    }

    public void export(List<Entity> entityList) {
        entityList.forEach(entity -> entity.accept(this));
    }

    @Override
    public void visit(Entity entity) {
        Path entityPath = exportPath.resolve(entity.name());
        Path parentPath = entityPath.getParent();
        try {
            Files.createDirectories(parentPath);
            entityPath = FileUtils.getFreePath(entityPath);

            try (OutputStream os = Files.newOutputStream(entityPath)) {
                os.write(entity.bytes());
            }

            Files.setLastModifiedTime(
                entityPath,
                FileTime.from(entity.lastModifiedDate().toInstant())
            );
        }
        catch (IOException ex) {
            throw new RuntimeException(ex);
        }
    }
}
```

ExportVisitor.java

```

public class P2PFileTransfer {
    private static final Logger logger = LogManager.getLogger(P2PFileTransfer.class.getSimpleName());
    protected static final ResourceManager resourceManager = ResourceManager.INSTANCE;
    private static final int DEFAULT_PORT = 44305;
    private final FileReceiver fileReceiver = new FileReceiver("received");
    private final FileSender fileSender = new FileSender();

    private static class FileSender {
        public void send(String address, Path path) throws IOException {
            try {
                InputStream is = Files.newInputStream(path);
                Socket connection = new Socket(address, DEFAULT_PORT);
                DataOutputStream dos = new DataOutputStream(connection.getOutputStream())
            } {
                String fileName = path.getFileName().toString();

                BasicFileAttributes bfa = Files.readAttributes(path, BasicFileAttributes.class,
LinkOption.NOFOLLOW_LINKS);
                long creationTime = bfa.creationTime().toMillis();
                long lastModifiedTime = bfa.lastModifiedTime().toMillis();

                dos.writeUTF(fileName);
                dos.writeLong(creationTime);
                dos.writeLong(lastModifiedTime);
                dos.writeLong(Files.size(path));

                IOUtils.copy(is, dos, 4096);
            }
        }
    }

    private record FileReceiver(String receiveDirName) {

        public void receive(Socket clientSocket) throws IOException {
            try {
                DataInputStream dis = new DataInputStream(clientSocket.getInputStream())
            } {
                Path receiveDir = initReceiveDir();
                receiveDir = initTodayDir(receiveDir);

                String fileName = dis.readUTF();
                Path filePath = initReceivedFile(dis, receiveDir, fileName);
                FileTime lastModifiedTime = FileTime.fromMillis(dis.readLong());

                long fileSize = dis.readLong();
                byte[] buffer = new byte[4096];
                try (OutputStream os = Files.newOutputStream(filePath, StandardOpenOption.WRITE)) {
                    int bytes;
                    while (fileSize > 0 && (bytes = dis.read(buffer, 0, (int) Math.min(buffer.length,
fileSize))) != -1) {
                        os.write(buffer, 0, bytes);
                        fileSize -= bytes;
                    }
                }
                Files.setAttribute(filePath, "basic:lastModifiedTime", lastModifiedTime,
LinkOption.NOFOLLOW_LINKS);
                logger.info(String.format(resourceManager.getString("serverSaved"), fileName,
filePath.toAbsolutePath()));
            }
        }

        private Path initReceiveDir() throws IOException {
            return Files.createDirectories(Path.of(receiveDirName));
        }

        private Path initTodayDir(Path receiveDir) throws IOException {
            SimpleDateFormat s = new SimpleDateFormat("yyyy-MM-dd");
            String dirName = s.format(new Date());
            Path todayDir = receiveDir.resolve(dirName);
            return Files.createDirectories(todayDir);
        }
    }
}

```



```

    }

    private Path initReceivedFile(DataInputStream dis, Path receiveDir, String fileName) throws
IOException {
        Path filePath = receiveDir.resolve(fileName);
        filePath = FileUtils.getFreePath(filePath);

        FileTime creationTime = FileTime.fromMillis(dis.readLong());

        Path result = Files.createFile(filePath);
        Files.setAttribute(filePath, "basic:creationTime", creationTime, LinkOption.NOFOLLOW_LINKS);

        return result;
    }

}

public Thread start() {
    Thread serverThread = new Thread() -> {
        logger.info(resourceManager.getString("serverStarted"));
        while (!Thread.interrupted()) {
            try {
                init();
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
        logger.info(resourceManager.getString("serverStopped"));
    };
    serverThread.start();
    return serverThread;
}

public void init() throws IOException {
    try (ServerSocket serverSocket = new ServerSocket(DEFAULT_PORT)) {
        logger.info(resourceManager.getString("serverWaiting"));
        serverSocket.setSoTimeout(15_000);
        try (Socket client = serverSocket.accept()) {
            logger.info(resourceManager.getString("serverConnected"));
            fileReceiver.receive(client);
        } catch (SocketTimeoutException ex) {
            logger.info(resourceManager.getString("serverTimeout"));
        }
    }
}

public void send(String address, Path path) throws IOException {
    logger.info(String.format(resourceManager.getString("clientSend"), path.getFileName()));
    fileSender.send(address, path);
}
}

```

P2PFileTransfer.java

```

public abstract class Environment {
    protected String inputType;
    protected final InputStream in;
    protected final PrintStream out;

    protected static final ResourceManager resourceManager = ResourceManager.INSTANCE;

    public Environment(InputStream in, PrintStream out) {
        this.in = in;
        this.out = out;
    }

    public void start() {
        Scanner scanner = new Scanner(in);

        String command;
        out.print(inputType);
        while (!(command = scanner.nextLine()).equals("exit")) {
            analyze(command);
            out.print(inputType);
        }
    }

    protected void incorrectCommand(String command) {
        if (!command.isBlank())
            out.printf(resourceManager.getString("incorrectCommand"), command);
    }

    protected abstract void analyze(String command);
}

```

Environment.java

```

public class MainEnvironment extends Environment {
    public MainEnvironment(InputStream is, PrintStream os) {
        super(is, os);
        this.inputType = "> ";
    }

    protected void analyze(String command) {
        switch (command) {
            case "help", "helper" -> new HelperEnvironment(in, out).start();
            case "archive", "archiver" -> new ArchiverEnvironment(in, out).start();
            case "fileTransfer", "transfer", "file transfer" -> new FileTransferEnvironment(in,
out).start();
            default -> incorrectCommand(command);
        }
    }
}

```

MainEnvironment.java

```
public class HelperEnvironment extends Environment {

    public HelperEnvironment(InputStream is, PrintStream os) {
        super(is, os);
        this.inputType = "helper> ";
    }

    protected void analyze(String command) {
        switch (command) {
            case "help", "helper" -> out.println("\n" + resourceManager.getString("helperHelp") + "\n");
            case "archive", "archiver" -> out.println("\n" + resourceManager.getString("helperArchiver") +
"\n");
            case "fileTransfer", "transfer", "file transfer" -> out.println("\n" +
resourceManager.getString("helperFileTransfer") + "\n");
            default -> incorrectCommand(command);
        }
    }
}
```

HelperEnvironment.java

```

@Getter
public class InputField {
    private final InputStream in;
    private final PrintStream out;
    private final String label;
    private final String name;
    private final Predicate<String> predicate;
    private final String errorMessage;
    private boolean isMultiple = false;
    private String splitter = "";

    public InputField(InputStream in, PrintStream out, String label, String name, Predicate<String>
predicate, String errorMessage) {
        this.in = in;
        this.out = out;
        this.label = label;
        this.name = name;
        this.predicate = predicate;
        this.errorMessage = errorMessage;
    }

    public InputField(InputStream in, PrintStream out, String label, String name, Predicate<String>
predicate, String errorMessage, boolean isMultiple, String splitter) {
        this(in, out, label, name, predicate, errorMessage);
        this.isMultiple = isMultiple;
        this.splitter = splitter;
    }

    public String start() {
        Scanner scanner = new Scanner(in);
        StringBuilder res = new StringBuilder();

        do {
            String currentInput;
            while (true) {
                out.print(label);
                currentInput = scanner.nextLine().trim();

                if (currentInput.isEmpty() && isMultiple) {
                    isMultiple = false;
                    res.deleteCharAt(res.length() - 1);
                    break;
                }

                if (predicate.test(currentInput)) {
                    res
                        .append(currentInput)
                        .append(splitter);
                    break;
                }
                out.println(errorMessage + "\n");
            }
        } while (isMultiple);

        return res.toString();
    }
}

```

InputField.java

```

public class InputForm {
    private final List<InputField> fields;
    private final Map<String, String> results = new HashMap<>();

    public InputForm(List<InputField> fields) {
        this.fields = fields;
    }

    public Map<String, String> execute() {
        for (InputField field: fields) {
            String res = field.start();
            results.put(field.getName(), res);
        }
        return results;
    }
}

```

InputForm.java

```

public class FormBuilder {
    private InputStream inputStream = System.in;
    private PrintStream printStream = System.out;
    private final List<InputField> fieldList = new LinkedList<>();
    public FormBuilder() {}

    public FormBuilder(InputStream inputStream, PrintStream printStream) {
        this.inputStream = inputStream;
        this.printStream = printStream;
    }

    public FormBuilder setInputStream(InputStream inputStream) {
        this.inputStream = inputStream;
        return this;
    }

    public FormBuilder setPrintStream(PrintStream printStream) {
        this.printStream = printStream;
        return this;
    }

    public FormBuilder addField(InputField field) {
        this.fieldList.add(field);
        return this;
    }

    public FormBuilder addField(String label, String name, Predicate<String> predicate, String
errorMessage) {
        this.fieldList.add(
            new InputField(inputStream, printStream, label, name, predicate, errorMessage)
        );
        return this;
    }

    public FormBuilder addField(String label, String name, Predicate<String> predicate, String
errorMessage, boolean isMultiple, String splitter) {
        this.fieldList.add(
            new InputField(inputStream, printStream, label, name, predicate, errorMessage, isMultiple,
splitter)
        );
        return this;
    }

    public InputForm build() {
        return new InputForm(fieldList);
    }
}

```

FormBuilder.java

```

public class ArchiverEnvironment extends Environment {

    public ArchiverEnvironment(InputStream is, PrintStream os) {
        super(is, os);
        this.inputType = "archiver> ";
    }

    protected void analyze(String command) {
        switch (command) {
            case "new" -> create();
            case "add" -> add();
            case "remove" -> remove();
            case "show" -> show();
            case "extract" -> extract();
            default -> out.println("incorrect command: " + command);
        }
    }

    private void create() {
        Map<String, String> res = new FormBuilder(in, out)
            .addField(resourceManager.getString("filenameLabel"), "name", InputPredicates.isFileName,
resourceManager.getString("filenameError"))
            .addField(resourceManager.getString("archiveTypeLabel"), "archType",
InputPredicates.isArchiveType, resourceManager.getString("archiveTypeError"))
            .addField(resourceManager.getString("compressionTypeLabel"), "compressType",
InputPredicates.isCompressorType, resourceManager.getString("compressTypeError"))
            .addField(resourceManager.getString("savePathLabel"), "path",
InputPredicates.isExistedPath, resourceManager.getString("existedPathError"))
            .addField(resourceManager.getString("fileToAddLabel"), "filePaths",
InputPredicates.isExistedPath, resourceManager.getString("existedPathError"), true, "|")
            .build()
            .execute();

        String archiveName;
        ArchiveType archType = getArchiveType(res.get("archType"));
        CompressorType compressType;
        if (res.get("compressType").isBlank()) {
            archiveName = res.get("name") + "." + archType.name().toLowerCase();
            compressType = null;
        } else {
            archiveName = res.get("name") + "." + archType.name().toLowerCase() + "." +
res.get("compressType").toLowerCase();
            compressType = getCompressorType(res.get("compressType"));
        }

        Path archivePath = FileUtils.getFreePath(Path.of(res.get("path"), archiveName));
        List<Path> filePaths = convertMultipleStringToPathList(res.get("filePaths"), "\\|");

        try {
            if (compressType != null) {
                getBasicArchiver(archType, compressType)
                    .create(archivePath, filePaths);
            } else {
                getBasicArchiver(archType)
                    .create(archivePath, filePaths);
            }
        } catch (IOException | ArchiveException e) {
            throw new RuntimeException(e);
        }

        out.printf("\n" + resourceManager.getString("successCreating") + "\n", archiveName, archivePath);
    }

    private void add() {
        Map<String, String> res = new FormBuilder(in, out)
            .addField(resourceManager.getString("archivePathLabel"), "path",
InputPredicates.isCorrectArchivePath, resourceManager.getString("archiveSupportError"))
            .addField(resourceManager.getString("fileToAddLabel"), "filePaths",
InputPredicates.isExistedPath, "Path is incorrect or file/directory does not exist!", true, "|")
            .build()
    }
}

```

```

        .execute();

    Path archivePath = Path.of(res.get("path"));
    Pair<ArchiveType, CompressorType> pairTypes = getArchiveAndCompressorTypesFromPath(archivePath);
    List<Path> filePaths = convertMultipleStringToPathList(res.get("filePaths"), "\\|");

    try {
        if (pairTypes.getRight() != null) {
            getBasicArchiver(pairTypes.getLeft(), pairTypes.getRight())
                .add(archivePath, filePaths);
        } else {
            getBasicArchiver(pairTypes.getLeft())
                .add(archivePath, filePaths);
        }
    } catch (IOException | ArchiveException e) {
        throw new RuntimeException(e);
    }

    out.printf("\n" + resourceManager.getString("successAdding") + "\n", archivePath.getFileName());
}

private void show() {
    Map<String, String> res = new FormBuilder(in, out)
        .addField(resourceManager.getString("archivePathLabel"), "path",
InputPredicates.isCorrectArchivePath, resourceManager.getString("archiveSupportError"))
        .build()
        .execute();

    Path archivePath = Path.of(res.get("path"));
    Pair<ArchiveType, CompressorType> pairTypes = getArchiveAndCompressorTypesFromPath(archivePath);

    List<Entity> entities;
    try {
        if (pairTypes.getRight() != null) {
            entities = getBasicArchiver(pairTypes.getLeft(), pairTypes.getRight())
                .read(archivePath, ArchiveReadingType.BASIC);
        } else {
            entities = getBasicArchiver(pairTypes.getLeft())
                .read(archivePath, ArchiveReadingType.BASIC);
        }
    } catch (IOException | ArchiveException e) {
        throw new RuntimeException(e);
    }

    out.printf("\n" + resourceManager.getString("showingListHeader") + "\n", archivePath);
    entities.stream()
        .map(Entity::name)
        .forEach(out::println);
    out.println();
}

private void remove() {
    Map<String, String> res = new FormBuilder(in, out)
        .addField(resourceManager.getString("archivePathLabel"), "path",
InputPredicates.isCorrectArchivePath, resourceManager.getString("archiveSupportError"))
        .addField(resourceManager.getString("entryToDeleteLabel"), "entryNames",
InputPredicates.isEntryName, resourceManager.getString("entryNameError"), true, "|")
        .build()
        .execute();

    Path archivePath = Path.of(res.get("path"));
    Pair<ArchiveType, CompressorType> pairTypes = getArchiveAndCompressorTypesFromPath(archivePath);
    List<String> entryNames = convertMultipleStringToStringList(res.get("entryNames"), "\\|");

    List<Entity> entities;
    try {
        if (pairTypes.getRight() != null) {
            entities = getBasicArchiver(pairTypes.getLeft(), pairTypes.getRight())
                .delete(archivePath, entryNames);
        } else {

```

```

        entities = getBasicArchiver(pairTypes.getLeft())
            .delete(archivePath, entryNames);
    }
} catch (IOException | ArchiveException e) {
    throw new RuntimeException(e);
}

out.println(resourceManager.getString("deletingListHeader"));
entities.stream()
    .map(Entity::name)
    .forEach(out::println);
out.println();
}

private void extract() {
    Map<String, String> res = new FormBuilder(in, out)
        .addField(resourceManager.getString("archivePathLabel"), "archivePath",
InputPredicates.isCorrectArchivePath, resourceManager.getString("archiveSupportError"))
        .addField(resourceManager.getString("extractPathLabel"), "extractPath",
InputPredicates.isExistedDirectory, resourceManager.getString("existedDirectoryError"))
        .build()
        .execute();

    Path archivePath = Path.of(res.get("archivePath"));
    Pair<ArchiveType, CompressorType> pairTypes = getArchiveAndCompressorTypesFromPath(archivePath);

    List<Entity> entities;
    try {
        if (pairTypes.getRight() != null) {
            entities = getBasicArchiver(pairTypes.getLeft(), pairTypes.getRight())
                .read(archivePath, ArchiveReadingType.FULL);
        } else {
            entities = getBasicArchiver(pairTypes.getLeft())
                .read(archivePath, ArchiveReadingType.FULL);
        }
    } catch (IOException | ArchiveException e) {
        throw new RuntimeException(e);
    }

    Path extractPath = Path.of(res.get("extractPath"));

    ExportVisitor visitor = new ExportVisitor(extractPath);
    visitor.export(entities);

    out.printf("\n" + resourceManager.getString("successExtracting") + "\n",
archivePath.getFileName(), extractPath);
}

private AbstractArchiver getBasicArchiver(ArchiveType archType) {
    return new BasicArchiverCreator().createArchiver(archType);
}

private AbstractArchiver getBasicArchiver(ArchiveType archType, CompressorType compressType) {
    return new BasicArchiverCreator().createArchiver(archType, compressType);
}

private List<Path> convertMultipleStringToPathList(String paths, String splitter) {
    return Arrays.stream(paths.split(splitter))
        .map(Path::of)
        .toList();
}

private List<String> convertMultipleStringToStringList(String paths, String splitter) {
    return Arrays.stream(paths.split(splitter))
        .toList();
}

private ArchiveType getArchiveType(String type) {
    return ArchiveType.valueOf(type.toUpperCase());
}

```



```

private CompressorType getCompressorType(String type) {
    return CompressorType.valueOf(type.toUpperCase());
}

private Pair<ArchiveType, CompressorType> getArchiveAndCompressorTypesFromPath(Path archivePath) {
    String fileExtension = FileNameUtils.getExtension(archivePath);
    ArchiveType archType;
    CompressorType compressType;
    if (InputPredicates.isArchiveType.test(fileExtension)) {
        archType = getArchiveType(fileExtension);
        compressType = null;
    } else {
        archType = getArchiveType(
            FileNameUtils.getExtension(
                FileNameUtils.getBaseName(archivePath)
            )
        );
        compressType = getCompressorType(fileExtension);
    }
    return new ImmutablePair<>(archType, compressType);
}
}

```

ArchiverEnvironment.java

```

public class FileTransferEnvironment extends Environment {
    private final P2PFileTransfer p2pFileTransfer = new P2PFileTransfer();

    public FileTransferEnvironment(InputStream in, PrintStream out) {
        super(in, out);
        this.inputType = "fileTransfer> ";
    }

    @Override
    protected void analyze(String command) {
        switch (command) {
            case "send" -> send();
            case "init" -> init();
            default -> incorrectCommand(command);
        }
    }

    private void send() {
        Map<String, String> res = new FormBuilder(in, out)
            .addField(resourceManager.getString("ipAddressLabel"), "ip", InputPredicates.isCorrectIp,
resourceManager.getString("ipAddressError"))
            .addField(resourceManager.getString("filePathLabel"), "filePath",
InputPredicates.isExistedFile, resourceManager.getString("existedFileError"))
            .build()
            .execute();

        Path filePath = Path.of(res.get("filePath"));
        try {
            out.println();
            p2pFileTransfer.send(res.get("ip"), filePath);
            out.println();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    private void init() {
        Thread server = p2pFileTransfer.start();

        out.println(resourceManager.getString("startStopping"));
        try {
            in.read();

            server.interrupt();
            out.println(resourceManager.getString("processStopping"));

            server.join();
        } catch (IOException | InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}

```

FileTransferEnvironment.java

```

public class InputPredicates {
    public static final Predicate<String> isFileName = s -> s.matches("[^/\\\\\\:;*?\\\"<>|]+");
    public static final Predicate<String> isEntryName = s -> s.matches("[^\\:;*?\\\"<>|]+");
    public static final Predicate<String> isArchiveType = s -> {
        try {
            ArchiveType.valueOf(s.toUpperCase());
            return true;
        } catch (IllegalArgumentException ex) {
            return false;
        }
    };

    public static final Predicate<String> isCompressorType = s -> {
        if (s.isBlank()) {
            return true;
        }
        try {
            CompressorType.valueOf(s.toUpperCase());
            return true;
        } catch (IllegalArgumentException ex) {
            return false;
        }
    };

    public static final Predicate<String> isExistedPath = s -> {
        try {
            Path p = Path.of(s);
            return Files.exists(p);
        } catch (InvalidPathException | NullPointerException ex) {
            return false;
        }
    };

    public static final Predicate<String> isExistedDirectory = s -> {
        try {
            Path p = Path.of(s);
            return Files.isDirectory(p);
        } catch (InvalidPathException | NullPointerException ex) {
            return false;
        }
    };

    public static final Predicate<String> isExistedFile = s -> {
        try {
            Path p = Path.of(s);
            return Files.isRegularFile(p);
        } catch (InvalidPathException | NullPointerException ex) {
            return false;
        }
    };

    public static final Predicate<String> isCorrectArchivePath = s -> isExistedPath.test(s) &&
(isArchiveType.test(FileNameUtils.getExtension(s)) ||
isCompressorType.test(FileNameUtils.getExtension(s)));

    public static final Predicate<String> isCorrectIp = s -> s.matches("(?:25[0-5]|2[0-4]
4)\\\\d|[01]?\\\\d\\\\d?)(\\\\.){3}(?:25[0-5]|2[0-4]\\\\d|[01]?\\\\d\\\\d?)$");
}

```

InputPredicates.java

```
public class ResourceManager {  
    private final String root = "location/lang";  
    private ResourceBundle currentBundle = ResourceBundle.getBundle(root);  
    public static final ResourceManager INSTANCE = new ResourceManager();  
    private ResourceManager() {}  
  
    public Locale getLocale() {  
        return currentBundle.getLocale();  
    }  
    public void changeResource(Locale locale) {  
        currentBundle = ResourceBundle.getBundle(root, locale);  
    }  
  
    public String getString(String key) {  
        return currentBundle.getString(key);  
    }  
}
```

ResourceManager.java

```

# Main
greeting = TRPZ-ARCHIVER\nCreator: Mishchuk Maksym Dmytrovych, IA-14

# All Environments
incorrectCommand = incorrect command: %s\n

# HelperEnvironment
helperHelp = \
    archive/archiver - to open the archiver environment;\n\
    file transfer/fileTransfer/transfer - to open the file transfer environment;\n\
    \n\
    Use that commands in the helper environment to see detail information.
helperArchiver = \
    new - to create a new archive;\n\
    add - to add files to the existing archive;\n\
    remove - to remove files from the existing archive;\n\
    show - to show a list of files of the existing archive;\n\
    extract - to extract files from the existing archive;
helperFileTransfer = \
    send - to send a file to the other client;\n\
    init - to initialize receiving files from other clients;

# ArchiverEnvironment
successCreating = Archive %s was successfully created in %s\n
successAdding = Archive %s was successfully added the files\n
successExtracting = Archive %s was successfully extracted to %s\n
showingListHeader = ----- Archive's files (%s) -----
deletingListHeader = ----- Successfully deleted -----

# FileTransferEnvironment
startStopping = ----- Press enter to stop the server -----
processStopping = ----- Process of stopping has started -----

# InputField (labels)
filenameLabel = Enter filename:
savePathLabel = Enter save path:
filePathLabel = Enter file path:
archivePathLabel = Enter archive path:
extractPathLabel = Enter extract path:
archiveTypeLabel = Enter archive type:
compressionTypeLabel = Enter compressor type (press enter if it's irrelevant):
fileToAddLabel = Enter file path to add (press enter to end):
entryToDeleteLabel = Enter entry name (press enter to end):
ipAddressLabel = Enter ip-address:

# InputFiled (errorMessages)
filenameError = Incorrect filename!
archiveTypeError = Incorrect archive type!
archiveSupportError = Path is incorrect or archive is not supported!
compressTypeError = Incorrect compress type!
existedPathError = Path is incorrect or file/directory does not exist!
existedDirectoryError = Path is incorrect or directory does not exist!
existedFileError = Path is incorrect or file does not exist!
entryNameError = Entry name is incorrect!
ipAddressError = Incorrect form of ip-address!

# P2PFileTransfer (logging)
serverStarted = Client server has been started
serverStopped = Client server has been stopped
serverWaiting = Client server is waiting for connection
serverConnected = Client server has got a connection
serverSaved = Client server has successfully saved file %s, path is: %s
serverTimeout = Client server has not got any connection
clientSend = Client has sent a file %s

```

lang.properties

```
public class FileUtils {  
    public static Path getFreePath(Path path) {  
        Path parentPath = path.getParent();  
  
        if (Files.exists(path)) {  
            String name = FileNameUtils.getBaseName(path);  
            StringBuilder extension = new StringBuilder();  
            extension.insert(0, FileNameUtils.getExtension(path));  
  
            while (!FileNameUtils.getExtension(name).isBlank()) {  
                extension.insert(0, FileNameUtils.getExtension(name) + ".");  
                name = FileNameUtils.getBaseName(name);  
            }  
  
            for (int i = 1; Files.exists(path); i++) {  
                path = parentPath.resolve(name + "(" + i + ")" + "." + extension);  
            }  
        }  
  
        return path;  
    }  
}
```

FileUtils.java

[Maxim-Mishchuk/trpz-archiver at develop \(github.com\)](https://github.com/Maxim-Mishchuk/trpz-archiver)

Penozumopiŭ

Додаток В

```
TRPZ-ARCHIVER  
Creator: Mishchuk Maksym Dmytrovych, IA-14  
  
> |
```

Запуск системи

```
TRPZ-ARCHIVER  
Creator: Mishchuk Maksym Dmytrovych, IA-14  
  
> helper  
helper> exit  
> archive  
archiver> exit  
> transfer  
fileTransfer> exit  
>  
> exit  
  
Process finished with exit code 0
```

Перехід між середовищами та вихід з програми

```

TRPZ-ARCHIVER
Creator: Mishchuk Maksym Dmytrovych, IA-14

> helper
helper> help

archive/archiver - to open the archiver environment;
file transfer/fileTransfer/transfer - to open the file transfer environment;

Use that commands in the helper environment to see detail information.

helper> archive

new - to create a new archive;
add - to add files to the existing archive;
remove - to remove files from the existing archive;
show - to show a list of files of the existing archive;
extract - to extract files from the existing archive;

helper> transfer

send - to send a file to the other client;
init - to initialize receiving files from other clients;

helper> |

```

Середовище помічника та його команди

```

TRPZ-ARCHIVER
Creator: Mishchuk Maksym Dmytrovych, IA-14

> archive
archiver> new
Enter filename: course_work
Enter archive type: zip
Enter compressor type (press enter if it's irrelevant):
Enter save path: F:\Office\Word\КНІ\3\ТРПЗ\курсова
Enter file path to add (press enter to end): F:\Code\Java\trpz\Archivator
Enter file path to add (press enter to end):

Archive course_work.zip was successfully created in F:\Office\Word\КНІ\3\ТРПЗ\курсова\course_work.zip

```

Середовище архіватора. Створення нового архіву


```

archiver> show
Enter archive path: F:\Office\Word\КПИ\3\ТППЗ\курсова\course_work.zip

----- Archive's files (F:\Office\Word\КПИ\3\ТППЗ\курсова\course_work.zip) -----
/Archivator/.git/COMMIT_EDITMSG
/Archivator/.git/config
/Archivator/.git/description
/Archivator/.git/FETCH_HEAD
/Archivator/.git/HEAD
/Archivator/.git/hooks/applypatch-msg.sample
/Archivator/.git/hooks/commit-msg.sample
/Archivator/.git/hooks/fsmonitor-watchman.sample
/Archivator/.git/hooks/post-update.sample
/Archivator/.git/hooks/pre-applypatch.sample
/Archivator/.git/hooks/pre-commit.sample
/Archivator/.git/hooks/pre-merge-commit.sample
/Archivator/.git/hooks/pre-push.sample
/Archivator/.git/hooks/pre-rebase.sample
/Archivator/.git/hooks/pre-receive.sample
/Archivator/.git/hooks/prepare-commit-msg.sample
/Archivator/.git/hooks/push-to-checkout.sample
/Archivator/.git/hooks/update.sample
/Archivator/.git/index
/Archivator/.git/info/exclude
/Archivator/.git/logs/HEAD
/Archivator/.git/logs/refs/heads/develop
/Archivator/.git/logs/refs/heads/developPullRequest
/Archivator/.git/logs/refs/heads/developUI
/Archivator/.git/logs/refs/heads/example
/Archivator/.git/logs/refs/heads/master
/Archivator/.git/logs/refs/remotes/origin/develop
/Archivator/.git/logs/refs/remotes/origin/developPullRequest
/Archivator/.git/logs/refs/remotes/origin/developUI
/Archivator/.git/logs/refs/remotes/origin/example
/Archivator/.git/logs/refs/remotes/origin/master
/Archivator/.git/logs/refs/stash

```

Середовище архіватора. Перегляд сутностей архіву (показано неповний список)

```
archiver> show
Enter archive path: E:\arch.zip

----- Archive's files (E:\arch.zip) -----
file1.txt
file2.txt
file3.txt

archiver> add
Enter archive path: E:\arch.zip
Enter file path to add (press enter to end): E:\export\dir
Enter file path to add (press enter to end):

Archive arch.zip was successfully added the files

archiver> show
Enter archive path: E:\arch.zip

----- Archive's files (E:\arch.zip) -----
file1.txt
file2.txt
file3.txt
/dir/file_dir2.txt

archiver>
```

Середовище архіватора. Додавання нових файлів до архіву

```
archiver> show
Enter archive path: E:\arch.zip

----- Archive's files (E:\arch.zip) -----
file1.txt
file2.txt
file3.txt
/dir/file_dir2.txt

archiver> remove
Enter archive path: E:\arch.zip
Enter entry name (press enter to end): file1.txt
Enter entry name (press enter to end): file3.txt
Enter entry name (press enter to end):









----- Successfully deleted -----
file1.txt
file3.txt

archiver> show
Enter archive path: E:\arch.zip

----- Archive's files (E:\arch.zip) -----
file2.txt
/dir/file_dir2.txt

archiver> |
```

Середовище архіватора. Видалення наявних сутностей з архіву

 arch.zip	28.12.2023 09:54	WinRAR ZIP archive	1 КБ
 test.zip	27.12.2023 19:36	WinRAR ZIP archive	1 КБ
 file1.txt	02.12.2023 16:41	Текстовый докум...	1 КБ
 file2.txt	02.12.2023 16:41	Текстовый докум...	1 КБ
 file3.txt	02.12.2023 16:41	Текстовый докум...	1 КБ
 export	26.12.2023 13:14	Папка с файлами	
 Programs	13.12.2023 00:01	Папка с файлами	
 Games	12.08.2023 18:46	Папка с файлами	

Розпакування архіву. Директорія до розпакування

```

archiver> show
Enter archive path: E:\arch.zip

----- Archive's files (E:\arch.zip) -----
file2.txt
/dir/file_dir2.txt











archiver> extract
Enter archive path: E:\arch.zip
Enter extract path: E:\

Archive arch.zip was successfully extracted to E:\

archiver>

```

Середовище архіватора. Процес розпакування

 arch.zip	28.12.2023 09:54	WinRAR ZIP archive	1 КБ
 test.zip	27.12.2023 19:36	WinRAR ZIP archive	1 КБ
 file1.txt	02.12.2023 16:41	Текстовый докум...	1 КБ
 file2(1).txt	02.12.2023 16:41	Текстовый докум...	1 КБ
 file2.txt	02.12.2023 16:41	Текстовый докум...	1 КБ
 file3.txt	02.12.2023 16:41	Текстовый докум...	1 КБ
 dir	28.12.2023 09:57	Папка с файлами	
 export	26.12.2023 13:14	Папка с файлами	
 Programs	13.12.2023 00:01	Папка с файлами	
 Games	12.08.2023 18:46	Папка с файлами	

Розпакування архіву. Директорія після розпакування

```

TRPZ-ARCHIVER
Creator: Mishchuk Maksym Dmytrovych, IA-14

> transfer
fileTransfer> send
Enter ip-address: 192.168.1.90
Enter file path: E:\arch.zip

28.12.2023 10:09:08.331 [main] INFO  P2PFileTransfer - Client has sent a file arch.zip

```

*Середовище файлообмінювача. Надсилання архіву
(вигляд зі сторони відправника)*

```

fileTransfer> exit
----- Press enter to stop the server -----
28.12.2023 10:08:35.000 [Thread-0] INFO  P2PFileTransfer - Client server has been started
28.12.2023 10:08:35.003 [Thread-0] INFO  P2PFileTransfer - Client server is waiting for connection
28.12.2023 10:08:50.015 [Thread-0] INFO  P2PFileTransfer - Client server does not get any connection
28.12.2023 10:08:50.018 [Thread-0] INFO  P2PFileTransfer - Client server is waiting for connection
28.12.2023 10:09:05.025 [Thread-0] INFO  P2PFileTransfer - Client server does not get any connection
28.12.2023 10:09:05.027 [Thread-0] INFO  P2PFileTransfer - Client server is waiting for connection
28.12.2023 10:09:06.270 [Thread-0] INFO  P2PFileTransfer - Client server has got a connection
28.12.2023 10:09:06.342 [Thread-0] INFO  P2PFileTransfer - Client server successfully saved file arch.zip, path: C:\Users\Я\Projects_Java\Projects_Java\trpz-archiver\received\2023-12-28\arch.zip
28.12.2023 10:09:06.343 [Thread-0] INFO  P2PFileTransfer - Client server is waiting for connection
28.12.2023 10:09:21.376 [Thread-0] INFO  P2PFileTransfer - Client server does not get any connection
28.12.2023 10:09:21.378 [Thread-0] INFO  P2PFileTransfer - Client server is waiting for connection

----- Process of stopping is started -----
28.12.2023 10:09:36.386 [Thread-0] INFO  P2PFileTransfer - Client server does not get any connection
28.12.2023 10:09:36.389 [Thread-0] INFO  P2PFileTransfer - Client server has been stopped
fileTransfer> fileTransfer>

```

*Середовище файлообмінювача. Ініціалізація серверу, прийняття та обробка запиту
(вигляд зі сторони приймача)*

```

----- Process of stopping is started -----
28.12.2023 10:09:36.386 [Thread-0] INFO  P2PFileTransfer - Client server does not get any connection
28.12.2023 10:09:36.389 [Thread-0] INFO  P2PFileTransfer - Client server has been stopped
fileTransfer> fileTransfer> exit
> archiver
archiver> show
Enter archive path: C:\Users\Я\Projects_Java\Projects_Java\trpz-archiver\received\2023-12-28\arch.zip

----- Archive's files (C:\Users\Я\Projects_Java\Projects_Java\trpz-archiver\received\2023-12-28\arch.zip) -----
file2.txt
/dir/file_dir2.txt

archiver>

```

Перегляд отриманого архіву