Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

Пояснювальна записка

до дипломної роботи
магістр
(освітньо-кваліфікаційний рівень)

на тему: **«МЕТОД ТА КРОСПЛАТФОРМЕНИЙ ЗАСІБ АРХІВАЦІЇ ОДНОТИПНИХ ФАЙЛІВ»**

Виконав: студент 2 курсу, групи 1КІ-18м
спеціальності
123 – Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)

Чирва П. В.
(прізвище та ініціали)
Керівник доц. каф. ОТ, к.т.н. Савицька Л. А.
(прізвище та ініціали)
Рецензент проф. каф. МБІС, д.т.н, Яремчук Ю. Є.
(прізвище та ініціали)

м. Вінниця - 2019 рік

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень магістр
Напрям підготовки 12 – Інформаційні технології
Спеціальність 123 – Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри Т. Б. Мартинюк

_____
"____"
_____20___року

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Чирві Павлу Васильовичу

1. Тема проекту (роботи) Метод та кросплатформений засіб архівації однотипних файлів
Керівник проекту (роботи) Савицька Людмила Анатоліївна, к.т.н., доц. каф. ОТ
затверджені наказом вищого навчального закладу від
"___"_____20__року №___
2. Строк подання студентом проекту (роботи)_____
3. Вихідні дані до проекту (роботи) список технічної літератури, аналіз, вивчення та дослідження процесів архівації даних в галузі інформаційних технологій, технічне завдання на магістерську роботу.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Аналіз сучасних методів та засобів архівації даних в галузі інформаційних технологій. Аналіз підходів, методів та моделей архівації даних. Методи стиснення та архівації даних та їх порівняльна характеристика. Огляд сучасних засобів архівації даних. Процеси архівації за кросплатформеною технологією Java. Метод архівації однотипних файлів. Процесу формування початкового та основного словника. Процес архівації. Розробка алгоритму роботи кросплатформеного засобу архівації однотипних файлів. Побудова користувацького інтерфейса для кросплатформеного засобу архівації однотипних файлів.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Порівняння алгоримів архівації та стиснення у випадку роботи з однотипними даними. Загальна схема методу архівації однотипних файлів. Процес формування словника. Діаграма процесу архівації. Блок-схема алгоритму роботи кросплатформеної програми архівації однотипних файлів. Алгоритм створення основного словника. Дизайн користувацького інтерфейсу.

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Технічний розділ	Савицька Л. А.		
Економічний розділ	Глущенко Л. Д.		

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
	Огляд існуючих підходів до розв’язання задачі архівації даних в галузі інформаційних технологій		
	Аналіз сучасних методів та засобів архівації файлів		
	Узагальнення інформації		
	Розробка методу та програмної реалізації архівації однотипних файлів		
	Узагальнення інформації		
	Проведення економічних розрахунків		
	Оформлення пояснювальної записки до дипломної роботи		
	Оформлення додатків та графічного матеріалу		

Студент _____ Чирва П. В.

(підпис) (прізвище та ініціали)

Керівник роботи _____ Савицька Л. А.

(підпис) (прізвище та ініціали)

РЕФЕРАТ

Дана магістерська кваліфікаційна робота присвячена розробці методу та кросплатформеній реалізації засобу архівації однотипних файлів. Цей програмний засіб дозволить ефективно виконувати архівацію великої кількості однотипних файлів.

Високий рівень вирішення поставленої задачі досягнуто за рахунок використання сучасної мови програмування Java.

В даній магістерській роботі виконано дослідження і аналіз сучасних методів та засобів архівації даних в галузі інформаційних технологій, аналіз підходів, методів та моделей архівації даних. Розглянуті методи стиснення та архівації даних та їх порівняльна характеристика. Виконано огляд сучасних засобів архівації даних, а саме, процеси архівації за кросплатформеною технологією Java.

Зокрема, у роботі розроблено метод архівації однотипних файлів, вдосконалено процес формування основного словника, вдосконалено процес архівації файлів.

REVIEW

This master's qualification work is devoted to the development of a method and cross-platform implementation of a file archiving tool of the same type. This software tool will allow you to efficiently archive a large number of files of the same type.

The high level of the solution to this task was achieved through the use of modern Java programming language.

In this master's thesis research and analysis of modern methods and means of archiving of data in the field of information technologies, analysis of approaches, methods and models of data archiving have been performed. The methods of compression and archiving of data and their comparative characteristics are considered. An overview of modern data archiving tools is performed, namely, processes of archiving on a cross-platform Java technology.

In particular, the method of archiving of the same type files has been developed, the process of formation of the main vocabulary has been improved, the file archiving process has been improved.

Зміст

ВСТУП

Актуальність теми дослідження. Задача компактного зберігання, перетворення та передавання інформаційних даних завжди була актуальною в галузі інформаційних технологій.

Інформаційні ресурси нині є продуктом інтелектуальної діяльності дійсно найбільш кваліфікованої й творчо активної частини молоді та працездатного населення світу. В останній чверті XX століття набуті інформаційні ресурси досягли (і продовжують досягати) настільки рекордних обсягів, що цілком повсякденним стали поняття «інформаційного вибуху», «інформаційної революції». В якості доказу є об'єктивне збільшення інформаційного потоку з початку цього сторіччя більш ніж в 30 разів! [1-2]. Отже, **актуальною** є наукова задача розробки та застосування принципово нових методів і засобів сприйняття, передачі, обробки, зберігання і розповсюдження інформаційних даних, таких, що здатних оперувати великими масивами інформації, причому, у реальному часі.

З метою забезпечити надійне збереження інформації створюють резервні копії даних. Задача збереження резервних копій у компактному вигляді є основою для процесів архівації та стиснення даних. В загальному випадку, основний зміст архівації полягає у створенні таких резервних копій, які потребували би значно меншого обсягу на інформаційних ресурсах, ніж та сама інформація у вихідному стані. Таким чином, в контексті під архівацією слід розуміти процес перекодування деякої сукупності файлів з метою зменшення загального об'єму пам'яті, який вони займають. Часто архівацією ще називають процес стиснення даних [3-5].

Нині відомо досить багато різних підходів до процесу архівації. Усі підходи мають в своїй основі різні підходи та різні методи, проте подібні вони в одному – це те, що вони сповідують принцип заміни рівномірного двійкового коду на нерівномірний. З метою архівації файлів та папок

використовують спеціальні програмні засоби, які називають архіваторами. Стиснуті файли поміщають у файли, який називають архівами [6-8].

Перші прототипи архіваторів з'явилися у 80-х роках минулого сторіччя. Основними можливостями сучасних архіваторів є такі [9]:

- занесення груп файлів та (або) підкаталогів в архів;
- можливість поновлення архіву;
- перегляд файлів з меж архіву;
- вилучення окремих файлів з архіву;
- захист файлів від несанкціонованого доступу (НСД);
- перевірка архіву на цілісність;
- створення багатотомних архівів;
- можливість створення архівів, що автоматично відкриваються.

Можливості сучасних програм-архіваторів дозволяють зекономити від 20 до 90 відсотків дискового простору. Файлом, що знаходиться в архіві, можна скористатися після того, як він буде відновлений у початковому вигляді, тобто розархівований (розпакований). Розархівування виконують або ті ж самі програми-архіватори в зворотному напрямку, або окремі програми, які називають розрахіваторами, серед яких найбільш відомими є: ZIP, JAR, RAR. Під час вибору конкретного засобу для архівування (розархівування) користувачі керуються багатьма критеріями, як то швидкістю роботи, коефіцієнти стискування даних, інтерфейс, сумісність тощо. Важливим є те, що для одного типу файлів кращим може бути один архіватор, а для іншого – інший [10-13].

Зв'язок роботи з науковими програмами, планами, темами. Магістерська робота виконана відповідно до напрямку наукових досліджень кафедри обчислювальної техніки Вінницького національного технічного університету; у рамках Національної стратегії розвитку освіти в Україні на період до 2021 р. (наказ Президента № 334/2012 від 25.06.2013 р.); плану наукової та навчально-методичної роботи кафедри обчислювальної техніки.

Мета та завдання дослідження. Метою дослідження магістерської кваліфікаційної роботи є збільшення середнього значення процесу архівації для великої кількості однотипних файлів.

Для досягнення поставленої мети необхідно виконати такі завдання:

провести аналіз сучасних методів та засобів архівації даних в галузі інформаційних технологій, виконати їх порівняльну характеристику та сформувати вимоги та обрати й обґрунтувати вибір методу, що задовольняв би меті даної магістерської кваліфікаційної роботи;

розглянути існуючі способи архівації за кросплатформеною технологією Java;

запропонувати метод архівації однотипних файлів згідно мети магістерської кваліфікаційної роботи, розробити ключові процеси роботи методу архівації однотипних файлів та виконати програмну реалізацію запропонованого методу архівації однотипних файлів;

провести тестування програмного продукту та виконати аналіз отриманих результатів.

Об'єкт дослідження – це сучасні процеси архівування та стиснення інформаційних даних.

Предмет дослідження – це методи та програмні засоби словникового методу архівування та стиснення інформаційних даних.

Методи дослідження. Дослідження, виконані під час роботи над кваліфікаційною магістерською роботою, ґрунтуються на теоретико-множинних підходах і принципах кросплатформеного підходу для виконання процесів архівації за кросплатформеною технологією Java; структурному проектуванні програмного забезпечення – для реалізації кросплатформеного програмного забезпечення архівації однотипних файлів; методах об'єктно-орієнтованого програмування – для реалізації алгоритмів і процесів методу архівації однотипних файлів та розробки відповідного програмного забезпечення.

Наукова новизна одержаних результатів полягає в такому:

вперше запропоновано метод архівації однотипних файлів, який дозволяє збільшити середнє значення процесу архівації однотипних файлів;

вдосконалено процес формування основного словника за рахунок застосування методу «ковзного вікна» до його формування;

вдосконалено процес архівації за рахунок застосування вдосконаленого процесу формування основного словника.

Практичне значення одержаних результатів полягає у такому:

1) Розроблено новий метод архівації однотипних файлів.
2) Вдосконалено процес формування основного словника за рахунок застосування до його формування методу «ковзного вікна».
3) Розроблено алгоритм роботи формування основного словника однотипних файлів, який дозволяє ефективніше стискати велику кількість однотипних файлів.
4) Розроблено програмний засіб для архівації однотипних файлів.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується строгістю постановки задач, коректним застосуванням методів та інформаційних технологій під час доведення наукових положень, строгим виведенням аналітичних співвідношень, порівнянням результатів з відомими, та збіжністю результатів дослідження з результатами, що отримані під час впровадження розробленого програмного засобу.

Особистий внесок магістранта. Усі результати, наведені у магістерській кваліфікаційній роботі, отримані самостійно[14].

1 АНАЛІЗ СУЧАСНИХ МЕТОДІВ ТА ЗАСОБІВ АРХІВАЦІЇ ДАНИХ

1.1 Сучасні методи архівації даних в галузі інформаційних технологій

Характерною особливістю усіх наявних типів інформаційних даних є їх надлишковість. Для людини, як для суб'єкта, надлишковість цих даних часто пов'язана із якістю отриманої даних, оскільки така надлишковість покращує нам зрозумілість та сприйняття цієї даних. Проте, коли ми говоримо про зберігання та передавання даних засобами обчислювальної техніки, то наявність надлишковості відіграє дуже негативну роль, оскільки вона призводить до зростання вартості зберігання та передавання даних [15].

Особливо актуальною є ця задача у випадках необхідності оброблення величезних обсягів даних при незначних чи недостатніх об'ємах носіїв даних. У зв'язку із цим постійно виникає задача відійти від надлишковості або процесу стиснення та архівації даних.

Базовий принцип, що є основою для процесу стиснення та архівації даних, полягає в економічному описанні повідомлення, згідно якого можливе відновлення його початкового значення із похибкою, яка контролюється [16].

Основоположником науки про процеси стиснення та архівації даних є Клод Шеннона. Його олноіменна теорема про оптимальне кодування висвітлює, що варто досягти під час кодування даних і на скільки та чи інша інформація при цьому стиснеться. Додатково ним були виконані дослідження за емпіричною оцінкою надлишковості (надлишковості) англійського тексту. Він пропонував піддослідним учням вгадувати наступну букву і оцінював ймовірність вірного вгадування.

Під час дослідів він дійшов висновку, що кількість даних в тексті, що написаний англійською мовою, коливається в межах 0.6-1.3 біта на символ. Незважаючи на те, що результати дослідів Шеннона були дійсно актуальні лише через десятиріччя, важко переоцінити їх значення. Перші алгоритми для процесів стиснення та архівації були примітивними у зв'язку із тим, що була досить примітивною сама обчислювальна техніка. Із розвитком потужностей комп'ютерів, стали можливими все більш потужні та досконалі алгоритми [17].

Одним із перших ттаких алгоритмів ефективного кодування даних був запропонований Д. А. Хаффманом в 1952 році. Ідея його алгоритму полягає в

такому: «знаючи ймовірності символів у деякому повідомленні, можна цілком описати процедуру побудови кодів змінної довжини, які складаються із цілої (цілочисельне число) кількості бітів». При чому, символам із більшою (вищою ступінню) ймовірністю ставляться у відповідність коротшим кодам.

У загальному випадку, коди Хаффмана мають властивість також префіксності (це значить, що жодне деяке КС не є префіксом іншого), і це дозволяє нам однозначно їх розкодувати. В той же самий час, класичний алгоритм Хаффмана на своєму вході отримує таблицю частот «зустрічальності» символів у нашому повідомленні. Далі на підставі такої таблиці частот і будується «дерево кодування». Нині алгоритм Хаффмана та його різноманітні модифікації широко застосовується під час процесів стиснення та архівації фото- і відеозображень (це JPEG, стандарти процесів стиснення та архівації MPEG), а такод, в архіваторах (RAR, PKZIP, LZH) [18-21].

Безумовним проривом в сфері процесів стиснення та архівації даних стало винайдення Лемпелем і Зівом в 1977 році «словникових алгоритмів». До цього моменту процеси стиснення та архівації зводилися у загальному випадку до примітивного кодування окремих символів. Винайдені ними «словникові алгоритми» дозволяли кодувати цілі рядки символів, що повторювалися, і таикй підхід дозволив досить різко підвищити ступінь ефективності процесів стиснення та архівації . Важливу роль при цьому зіграв винахід арифметичного кодування, і тепер – це дозволило реалізувати ідею Шеннона про оптимальне кодування.

Величним по-справжньому, наступним таким проривом став винахід в 1984 р. алгоритму PPM. І він довго залишався непоміченим, бо вимагав дуже великих ресурсів пам’яті, і це було серйозною проблемою на той час.

Також, у 1984 р. побачив світ алгоритм LZW і став відомим завдяки простоті, рекламі і невимогливості до ресурсів, при цьому, мав досить низький ступінь процесу стиснення та архівації .

Нині алгоритм PPM є найкращим алгоритмом для процесу стиснення та архівації текстовоих даних, а LZW вже не застосовується в нових аплікаціях (але!!! Широко застосовується старими).

Сучасні алгоритми стиснення та архівації вже впритул наблизилися до Шеннонівської оцінки 1.3 біта на символ, і науковці вважають обчислювальну техніку спроможною виконати це завдання [22].

З метою досягти потужних резкльтатів роботи процесу стиснення та архівації треба використовувати складніші методи та алгоритми. Безумовно, є швидкі реалізації PPM для текстових даних і SPIHT для графічних, при цьому, мають досить високий ступінь процесу стиснення та архівації. Можна сказати, що майбутнє за такими алгоритмами із високими вимогами до ресурсів і постійно зростаючою ступінню ефективності процесів стиснення та архівації .

Алгоритми, що працюють для процесів стиснення та архівації можуть підвищувати свою ефективність процесів збереження і передавання даних, в першу чергу, за допомогою зменшення ступеню їх надлишковості. Алгоритми процесів стиснення та архівації використовують на вході текст джерела і видають відповідний стиснутий текст, тоді як зворотній алгоритм має на своєму вході стиснутий текст і отримує із нього на своєму виході початковий текст.

Більшість алгоритмів процесів стиснення та архівації розглядають вихідний текст як набір рядків, що складаються із літер алфавіту вихідного тексту. Надлишковість в подані рядка S є такою (1.1):

$$L(S) - H(S), \qquad (1.1)$$

де L(S) – це довжина повідомлення на вході в бітах,

H (S) – ентропія, тобто, ступінь змісту даних, також виражена в бітах.

Таких методів чи алгоритмів, які могли б без втрати даних стиснути рядок даних до ще меншого кількості бітів, ніж складає його ентропія, не існує. Задача економного кодування даних в системах управління базами даних (УБД) досі є актуальним. Науковці говорять про недостатній теоретичний розгляд задачі і неефективність підтримки процесів стиснення та архівації даних в промислових системах управління базами даних (СУБД) [23]. Використання в СУБД таого економного кодування без втрат даних

призводить до гарних результатів. Це зменшення фізичного розміру самої БД, журнальних, архівних її файлів, також, відбувається зниження вимог до обсягу оперативної пам’яті (ОП).

Ефективна реалізація підтримки процесів стиснення та архівації даних значно покращує якість СУБД. Інтерес до задачі процесів стиснення та архівації даних саме в СУБД був зумовлений бажанням зменшити фізичний обсяг БД. Вартість підсистем введення-виведення становила основну частку вартості апаратури. Під час інтегруванні в СУБД методів і процесів стиснення та архівації без втрат даних досягалася значна економія. Незначне збільшення кількості тактів CPU для процесів кодування-декодування компенсувалося зниженням витрат на обслуговування підсистем введення-виведення.

Результат застосування процесу стиснення та архівації актуальний і нині: значне зменшення ціни пристроїв зберігання даних супроводжувалося збільшенням розміру БД. Важливою є продуктивність системи керування даними за рахунок використання процесів стиснення та архівації . Важливо: економне кодування сприяє також криптографічному захисту.

Усунення статистичної надлишковості підвищує крипостійкість алгоритмів шифрування даних і є етапом під час проесів шифрування даних [58]. Використання процесів стиснення та архівації даних пов’язане із вирішенням багатьох завдань. Реалізація економного кодування пов’язана із компромісом між ступенем процесу стиснення та архівації , необхідним об’ємом ОП, кількістю звернень до зовнішньої пам’яті та ОП, обчислювальними витратами [30, 45].

Прикладом потреби в зберіганні і обробці великих об’ємів інформації є використання електронних бібліотек космічних даних. Такі інформаційні системи (ІС) використовують потоки супутникових даних для потреб дистанційного дослідження [16].

Скажімо, один кадр супутника в 6 спектральних каналах і 1 тепловому покриває площу 170 на 185 кв. км. Тоді обсяг одного кадру становить 400 Мбайт. Щоденні обсяги оцінюються в 480-490 Гбайт, а обсяг оброблених даних в системі EOSDIS досягає 1600 Гбайт на добу [17, 26].

Розвиток систем зберігання даних визначають цінність даних в науці та суспільстві, можливість доступу до них, а керування ними є умовою забезпечення бізнеспроцесів. За даними Gartner, серед компаній, постраждалих від природніх та техногенних катастроф і в результаті мають велику втрату корпоративних даних, 43% не змогли продовжити свою роботу.

1.2 Аналіз підходів, методів та моделей архівації даних

Методи стиснення та архівації даних можна розділити на два типи:

1) без спотворення (loseless) – методи стиснення та архівації гарантують, що декодовані дані будуть в збігатися із вихідними;
2) із втратами (lossy) – методи процесу стиснення та архівації можуть спотворювати вихідні дані, за рахунок видалення несуттєвих частин даних, після чого повне відновлення неможливе [18].

Методи стиснення та архівації даних без втрат засновані на усуненні надлишковості подання даних. Ефективність кодування досягається за рахунок подання малоймовірних подій більш довгими словами, ніж подій із вищою ймовірністю настання.

Якщо ймовірність настання події є деяке значення P, то, відповідно теоремі Шеннона, цю подію варто кодувати словом завдовжки $-\log_2 P$ біт. Методи стиснення та архівації даних явно або неявно мають в основі саме це [15].

У результаті процесів ефективного кодування «1» вихідних даних ставиться у відповідність КС (КС). КС складається із послідовності двійкових цифр. Сукупність кодових слів утворює сам код. У випадку, коли довжини всіх кодових слів сталі, то застосовується код має фіксовану (постійну) довжину, інакше – змінну. Якщо вихідні дані можуть бути відновлені по масиву кодових слів, то кодування не призводить до втрат даних.

Ефективність процесів стиснення та архівації визначається ступінню стиснення. Ступінь самого стиснення становить значення, яке дорівнює

відношенню обсягу вихідних даних до об'єму відповідних їм стиснутих даних і вимірюється в кількості разів.

Всі методи стиснення та архівації прийнято розділяти на 2 класи:

1) статистичного кодування
2) словникового стиснення та архівації [15].

У схемах процесів стиснення та архівації часто застосовуються допоміжні перетворення, що забезпечують та сприяють виконанню ефективного кодування.

1.3 Аналіз статистичного методу

Підходи статистичного кодування опираються на відому теорему Шеннона і містять 2 етапи:

1) оцінка ймовірності кодованих елементів (моделювання);
2) власне кодування.

На етапі 1) виконується заміщення визначеного елемента S_i із оцінкою ймовірності появи $q(S_i)$ деяким кодовим словом довжиною $-\log_2 q(S_i)$ бітів. Це ще називають ентропійним кодуванням. Оцінки ефективності можуть бути отримані із безумовних частот ступеню частоти зустрічання елементів, тобто із урахуванням контексту, більш складнішим способом. Відновлення вихідних даних без втрат забезпечується в тому випадках, коли кодувальник і декодувальник оперують тими ж самими оцінками в кожний $q(S_i)$ момент часу.

При цьому задача кодування деякого елементу із заданою ймовірністю вирішується за допомогою різновидів методу Хаффмана і арифметичних основ процесу стиснення та архівації [16].

Розглянемо алгоритм Хаффмана, що визначає процедуру побудови коду саме змінної довжини, де середня надлишковість мінімальна для неблочних кодів, тобто вони задають відображення лиш одного вихідного елемента в одне КС. Оскільки це слово може бути представленим лиш цілим числом біт, під час кодуванні за Хаффманом здійснюється наближення значення $q(S_i)$ дробами, що рівні степеню 2. Цей алгоритм є незастосовним для економного

кодування елементів двійкового алфавіту. Код Хаффмана є префіксним кодом і тому зображується у вигляді дерева. Тут застосовується двопрохідна схема (статистичний алгоритм Хаффмана): при 1 перегляді підраховується статистика зустрічальності елементів, тобто будується «модель даних», на підставі цієї моделі формується сам код; при 2 перегляді дані стискаються за допомогою отриманого коду. Оцінки ймовірності $q(S_i)$ під час кодування є сталими, і код теж є сталим. Зауважимо, що є адаптивний однопрохідний варіант цього алгоритму, але він володіє суттєво більшою обчислювальною складністю і на практиці не застосовується [17].

Арифметичне стиснення чи архівація, або ж арифметичне кодування, дозволяє під час кодування кількох елементів подавати кожен елемент в середньому дробовим числом бітів. Таким чином, арифметичне стиснення та архівація забезпечує більшу степінь стиснення, ніж кодування за Хаффманом. Блок кодованих елементів є дробом, що визначається добутком оцінок ймовірності $q(S_i)$ всіх елементів блоку. Чим значення $q(S_i)$ є меншим, тим довшим є дріб, і тоді потрібно більше двійкових символів для його подання. Алгоритм цього декодування є складнішим, ніж алгоритм методу Хаффмана, він дозволяє відновити вихідні дані дійсно без втрат. Арифметичне кодування, звісно, не вимагає явної перебудови коду під час змінення оцінок ймовірності, тому застосовується адаптивна однопрохідна схема, яка дозволяє натуральним чином враховувати локальні особливості даних.

1.4 Аналіз методу словникового стиснення та архівації

Суть процесу словникового стиснення та архівації полягає в заміні послідовностей деяких елементів вихідних даних на IDи (ID) таких фраз із надр деякого словника, що збігається із послідовністю що замінюється. Методи словникового стиснення та архівації використовують повторюваність рядків символів. Словник, як сукупність фраз будується по-різному. Скажімо, в нього можуть вклювходити чатися такі рядки та символьні послідовності, які є найбільш значимими і мають свої характеристики, де q(L-Li) – частота, з якою ми зустрічаємо послідовність, L – її довжина, Li – довжина ID (покажчика) фрази словника.

Серед словникових методів найбільше поширення мають методи Зіва-Лемпеля і їх можна розділити на 2 сімейства: LZ77 (LZ1) і LZ78 (LZ2). Усі схеми цього сімейства LZ77 базуються на одноіменному методі, що запропонований в 1977 році у відомій статті «Універсальний алгоритм для послідовного стиснення даних» [43, 59, 60]. У методах і схемах даного сімейства алгоритмів як словника є порція вже оброблених даних. Послідовність тоді кодується зазначенням позиції початку еквівалентної фрази в словнику (тобто, зсуву) і довжини збігу. Утоврена пара в такому випадку є вказівником. Якщо елемент (той елемент, що кодується) відсутній в словнику, то він якось позначається і подається в тому виглдяі, в якому він є. І тоді цей елемент називається літералом. Виходячи із способу формування словника бачимо, що методи даного сімейства LZ77 є адаптивними.

На практиці схеми типу LZ77 застосовуються спільно із алгоритмами статистичного кодування покажчиків (вказівників) і літералів. Скажімо, в методі LZH для економного кодування покажчиків (вказівників) і літералів застосовується відомий алгоритм Хаффмана.

Також, зазначимо, що в схемах сімейства LZ78 у сам словник поміщуються не всі-всі послідовності, ті, які зустрілися в даному обробленому масиві даних, а лише так звані «перспективні» із точки зору величини ймовірності їх появи в майбутньому. От, в методі LZ78 кожна нова фраза формується як злиття (конкатенація) однієї із фраз S словника, що має найдовший збіг із поточною закодованою послідовністю символів S*, і символу S. Символ S є символом, що слідує за послідовністю S*=S. На відміну від сімейства LZ77, в такому словнику не може бути просто фраз. У найвідомішого представника даного сімейства LZ78, тобто, у методі LZW, словник ініціалізується фразами для всіх символів алфавіту кодованих даних. Покажчики (вказівники) на ці фрази кодуються словами фіксованої довжини, і визначається розмірами самого словника. У межах методу сімейства LZ78 просто реалізувати ефективне неадаптивне і напівадаптивне стиснення та архівацію, під час яких словник будується заздалегідь [18].

Словник, що застосовується у таких словникових методах для стиснення та архівації даних, можна розглядати як аналог статистичної моделі, що і застосовується у статистичних методах.

1.5 Інші методи стиснення та архівації даних та їх порівняльна характеристика

1.5.1 Кодування серій, RLE

Поширеним методом стиснення та архівації є кодування довжин серій (Run Length Encoding, RLE). Запропонований метод дозволяє кодувати послідовності елементів. Існує велика кількість різновидів кодування довжин серій. Скажімо, послідовність ідентичних елементів може бути представлена 3 послідовностями: 1) «прапор використання кодування, 2) довжина серії, 3) повторюваний елемент» [15].

Для кодування цілих чисел із невідомою, але такою, що монотонно зменшується, ймовірністю, застосовуються «універсальні» коди. Надлишковість універсальних кодів цілих чисел для будь-якого розподілу і визначеної кодованого числа n є обмеженою зверху, якщо виконується умова (1.2):

$$p(n) \geq p(n+1). \tag{12}$$

Універсальними є, без сумніву, коди Елайеса [10, 48]. Програмні реалізації алгоритмів RLE є простими, мають високу швидкість, проте забезпечують недостатній рівень стиснення та архівації. Найкращими об’єктами, що нииим опреує алгоритм – це графічні файли, що мають досить великі ділянки одного кольору і кодуються, відповідно, довгими послідовностями однакових байтів. Проте, RLE не підходить для зображень із плавним переходом кольорів, типу фото. RLE однак може давати досить помітний виграш на деяких типах файлів БД, що мають, скажімо, таблиці із фіксованою довжиною полів. А от для текстових даних RLE не підходить [23].

Засобами RLE можуть бути заархівовані будь-які файли із двійковими даними, і це пов’язано з тим, що ці специфікації містять повторювані байти в ділянці вирівнювання даних. Сучасні системи стиснення та архівації частіше використовують алгоритми на основі LZ77, які є узагальненням методу RLE.

Звукові дані, що містять довгі послідовні серії байтів (це низькоякісні звукові приклади, samples) можуть бути ефективно стиснуті за допомогою RLE.

Серед позитивних сторін RLE є невимогливість до пам’яті під час роботи, швидке виконання задач. Алгоритм застосовується в таких форматах, як: PCX, TIFF, BMP.

1.5.2 Алгоритм LZ77

Цей словниковий алгоритм процесу стиснення та архівації є найстарішим серед методів LZ. Його описання було опубліковано в 1977 р., але сам алгоритм розроблено у 1975. Є основою сімейства «словникових схем» – алгоритмів із ковзаючим словником, або вікном. Дійсно, алгоритм LZ77 у якості словника застосовує блок закодованої послідовності. У відповідності з обробленням положення цього блоку щодо початку послідовності, яка постійно змінюється, словник ніби «ковзає» по вхідному потокові даних.

В 1977 Абрам Лемпель і Якоб Зів висунули ідею щодо формування «словника» загальних послідовностей даних. І з того часу алгоритми LZ* називаються методами стиснення та архівації із використанням «ковзного вікна». Ковзне вікно можна уявити у вигляді буфера, який запам’ятовує «почуту» раніше інформацію і надає до неї доступ. Таким чином, сам процес кодування за LZ77 нагадує написання програми, де команди звертаються до елементів «ковзаючого вікна», і дозволяють замість значень стисненої послідовності вставляти посилання на ці значення в «ковзному вікні». При цьому розмір ковзного вікна може динамічно змінюватися і складати відповідно 2, 4 або 32 Кб. Зазначимо, що розмір самого вікна кодувальника може бути меншим або дорівнювати розмірам вікна декодувальника, але при цьому, не навпаки [15-18].

Метод кодування згідно із означеним принципом ковзаючого вікна враховує дані, що вже зустрічалися, тобто ті, яка вже відомі для кодувальника і декодувальника (друге і кожна наступне входження деякого ряду символів замінюються посиланнями на її перше входження).

До недоліків даного методу можна віднести:

– довжина підрядка, яку можна закодувати, обмежена розміром буфера;

– невисока ефективність під час кодуванні незначного обсягу даних.

1.5.3 Алгоритм Шеннона-Фано

Алгоритм Шеннона-Фано – один із перших алгоритмів стиснення та архівації, що має велику схожість із алгоритмом Хаффмана. Він застосовує коди змінної довжини: символ, який часто зустрічається у повідомленні кодується деяким кодом меншої довжини, а той, що рідко зустрічається – кодом дещо більшої довжини. Коди Шеннона – Фано префіксні, і ніяке КС не є префіксом якогось іншого і це дозволяє однозначно декодувати послідовність. Цей алгоритм Шеннона-Фано застосовує надлишковість у формат повідомлення, у вигляді неоднорідного розподілу частот символів алфавіту, тобто замінює коди частіших символів короткими двійковими послідовностями, а коди більш рідкісних символів – довшими двійковими послідовностями.

Код Шеннона-Фано будується за допомогою дерева, так само, як і алгоритм Хаффмана. Побудова дерева починається з кореня, далі множина кодованих елементів відповідає так званій, вершині 1 рівня. Вона розбивається на 2 підмножини із приблизно проміжними сумарними ймовірностями, які, у свою чергу, відповідають 2 вершинам 2-го рівня, що з’єднані із коренем. Далі кожна із цих підмножин розбивається на 2 підмножини із приблизно проміжними сумарними ймовірностями. Їм відповідають вершини 3-го рівня. Якщо деяка підмножина містить 1 елемент, то йому відповідає 1 кінцева вершина кодового дерева і при цьому підмножина розбиттю не підлягає. І так відбувається доти, допоки не отримаємо всі кінцеві вершини. Гілки такого кодового дерева помічені символами 1 і 0, як у випадках коду Хаффмана [19].

Кодування за методом Шеннона-Фано є старим методом стиснення та архівації, і нині не застосовне, до того ж, більш ефективним вважається стиснення та архівація за методом Хаффмана.

1.5.4 Алгоритм Хаффмана

Алгоритм Хаффмана – це такий адаптивний алгоритмом оптимального префіксного кодування алфавіту із мінімальною надлишовістю і був розроблений в 1952 році аспірантом Девідом Хаффманом під час написання ним наукової роботи. Нині застосовується в програмах стиснення та архівації даних і складається із 2 основних етапів:

1) Побудова оптимального кодового дерева.
2) Побудова відображення код-символ на основі побудованого дерева.

Це один із перших алгоритмів ефективного кодування даних, ідея алгоритму така: знаючи ймовірності символів у повідомленні, можна описати деяку процедуру побудови кодів змінної довжини, що складаються із цілої кількості бітів. Символам із більшою ймовірністю ставляться у відповідність коротші коди. Коди Хаффмана, звісно, мають властивість префіксності (тобто жодне КС не є префіксом іншого), що й дозволяє однозначно їх декодувати. Класичний алгоритм Хаффмана на своєму вході має таблицю частот символів у повідомленні, на основі якої будується дерево кодування [20].

1) Символи вхідного алфавіту формують список вільних вузлів. Кожен вузол має вагу, який може дорівнювати або значенню ймовірності, або кількості входжень символу в повідомлення.
2) Далі виибираються 2 вільних вузла дерева із мінімальними вагами.
3) Створюється їх батько із такою вагою, що дорівнює їх сумі їх ваг.
4) Далі батько додається в список вільних вузлів, а 2 його нащадка видаляються із даного списку.
5) Першій дузі, котра виходить із батьківського вузла, ставиться у відповідність біт логічної одиниці«1», інший – логічного нуля «0».

6) Кроки, починаючи із №2, повторюються доти, допоки в списку вільних вузлів не залишиться тільки 1 вільний. Він надалі і вважатиметься коренем дерева.

На відміну від кодів Шеннона-Фано, коди Хаффмана залишається завжди оптимальними в тому числі для вторинних алфавітів із більш ніж 2 символами.

Класичний алгоритм Хаффмана має ряд таких недоліків [21].

1) З метою відновлення вмісту стиснутого повідомлення декодувальник має знати таблицю частот кодувальника.
2) Виходить, що довжина стиснутого повідомлення збільшується на значення довжини таблиці частот, що має надсилатися перед даними, і це нівелює стиснення та архівацію повідомлення.
3) З'являється необхідність повної частотної статистики перед початком процесу кодування і це потребує 2 проходів за повідомленням: першого – для побудови моделі повідомлення, другого – для власне кодування.

1.5.5 Алгоритм стиснення та архівації PPM

PPM (Prediction by Partial Matching – прогнозування за частковим співпадінням) – це адаптивний статистичний алгоритм стиснення та архівації даних без втрат, заснований на контекстному моделюванні і прогнозуванні. PPM застосовує контекст – множину символів в стисненому потоці, що йдуть перед даними, аби передбачати значення символу на основі статистики. Сама по собі модель PPM лише передбачає значення символу, а саме стиснення та архівація здійснюється за алгоритмом Хаффмана і арифметичним кодуванням [22].

Довжина контексту, який застосовується підчас прогнозу обмежена і позначається символом n, що визначає порядок моделі PPM(n). Необмежені моделі існують і позначаються як PPM*. Якщо передбачення символу за контексту із n символів не може бути виконаним, то відбувається спроба передбачити його за допомогою n-1 символів і так далі. Такий рекурсивний перехід до моделей із все меншим порядком відбувається допоки

прогнозування не справдиться в одній із моделей, або коли контекст стане 0 довжини (n = 0).

Важливою для PPM є задача оброблення нових символів, ще не зустрічалися у вхідному потоці. Це задача носить назву «нульової 0 частоти». Деякі варіанти реалізацій PPM мають лічильник нового символу рівним константі, скажімо, 1. Нинішні реалізації PPM є кращими серед алгоритмів процесу стиснення та архівації без втрат для текстів [23].

Варіанти алгоритму PPM нині момент активно застосовуються, для компресії надлишкових і текстових даних. Відомі реалізації алгоритму PPM: RAR, 7-Zip, WinZip.

В таблиці 2.1 наведено стисле порівняння розглянутих алгоритмів стиснення та архівації даних, а на рис. 2.1 – графічне оформлення досліження у випадку роботи з однотипними даними.

Таблиця 2.1 – Порівняння алгоритмів архівації та стиснення

Алгоритм	Ступінь стиснення та архівації	Стиснення без втрат?	Кількість проходів	Застосування	Належність
Статичний алгоритм Хаффмана	5-6	так	2	Стиснення текстів бінарної інформації, ін.	Неблоковий, статистичний
Адаптивний алгоритм Хаффмана	4-5	так	1	Стиснення текстів бінарної інформації, ін.	Неблоковий, статистичний
LZ88	9-10	так	1	Універсальений	Блоковий
PPM	10-12	так	1	Універсальний (краще – текст і зображення)	Адаптивний, ститастичний
RLE	2-6	так	1	Стиснення аудіо та зображень	Блоковий, статистичний

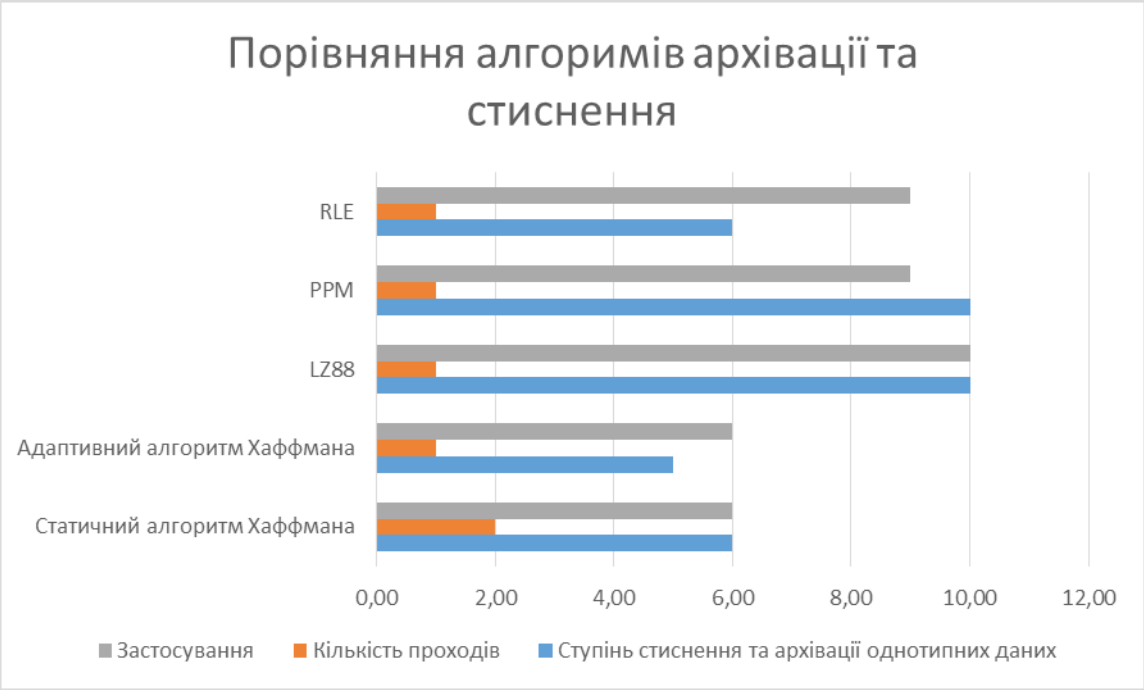


Рисунок 1.1 - Порівняння алгоримів архівації та стиснення у випадку роботи з однотипними даними

В загальному, найбільш ефективними методами стиснення у випадку роботи з однотипними даними стиснення та архівації даних є PPM та LZ. Для виконання поставленої задачі у даній роботі застосуємо алгритм LZ. Безумовноє, інші методи також мають право на існування, бо вони дуже часто застосовуються в інших задачах.

1.6 Огляд сучасних засобів архівації даних

Часто резервні копії даних доводиться робити при умовній обмеженості ресурсів розміщення цих даних, скажімо власникам персональних комп'ютерів. Тоді вони використовують програмну архівацію. У загальному, засоби для виконання процесів архівація – це виконання процесу злиття декількох файлів і навіть каталогів у єдиний спільний файл-архів, що дає нам водночас зі скороченням загального об'єму вихідних даних (шляхом усунення надлишковості, але без втрат даних) також можливість точного відновлення вихідних файлів. Діяльність більшості програмних засобів архівації заснована на використанні алгоритмів стиснення та архівації, що були запропоновані ще у 80-х рр. Абрахамом Лемпела і Якобом Зівом. Найбільш відомими і актуальними нині є такі архівні формати;

- ZIP, ARJ – застосовується переважно для операційних систем DO та Windows;
- TAR – застосовується виключно для спеціалізованої операційної системи Unix;
- Такоож представлений міжплатформений формат JAR (Java ARchive);
- Популярний формат RAR.

Користувачеві слід лише вибрати для вирішення своіх задач необхідний програмний засіб, що забезпечив би роботу із обраним форматом архівування. Зробити це можна шляхом оцінки всіх характеристик, як то:

- швидкодії;
- ступеню стиснення та архівації ;
- сумісності із необхідною кількістю форматів;
- зручності інтерфейсу;
- вибору операційної системи, тощо.

Список таких програм дуже великий – PKZIP, PKUNZIP, ARJ, RAR, WinZip, WinArj, ZipMagic, WinRar тощо. Більшість із цих програмних засобів архівування не варто купувати, оскільки вони, як правило, йдуть в поставках пакетів прикладних прорам, як програми умовно-безкоштовні або вільного поширення.

1.6.1 Сучасні засоби архівації даних у Windows

У системі Windows не передбачено вбудованих функцій для роботи із упакованими архівами, тому є сенс інсталювати спеціалізовані утиліти і таким чином закрити задачу різноманіття архівних форматів. Місткість носіїв і обсяг оперативної пам’яті постійно зростає, канали передавання даних стають усе більш потужними, та все ж об’єми даних, що передаються є потужними і архівування є досі актуальною задачею.

Прийнято розрізняти архівацію і стиснення даних. Архівація – це злиття декількох файлів і каталогів в єдиний файл під назвою «архів» (використання такої технології в чистому вигляді - це формат TAR). А коли говоримо про стиснення – мова йде про скорочення обсягу початкових файлів шляхом усунення надлишковості. Сучасні архіватори і стискують і архівують

одночасно, але є і чисто однонаправлені утиліти типу Gzip, що стискують окремі файли, перетворюючи їх у формат Z або GZ.

Під час вибору інструментів для роботи із упакованими даними й архівами варто враховувати 2 чинники:

1) ефективність – баланс між економією та продуктивністю роботи,
2) сумісність – обмін даними із іншими користувачами.

Сумісність в пріоритеті, оскільки за ступенем стиснення конкуруючі формати і інструменти різняться на відсотки (не в рази), а обчислювальна потужність сучасних машин робить час оброблення архівів некритичним. Тому під час вибору інструменту для роботи із архівом йдеться більше про сумісність.

Дія більшості засобів процеси стиснення засновані на використанні алгоритмів стиснення, запропонованих ще в 80-х рр. Абрамом Лемпелем і Якобом Зивом. Велика кількість популярних архівних форматів, типу ZIP, LZH, ARJ, ARC, ICE та інші – з’явилися за часів DOS. Тоді використовувалися спеціалізовані архіватори-пакувальники, наприклад, утиліти PKZip/PKUnzip, LHA, Arj, які дозволяли архівувати папки і забезпечували високу ступінь стиснення для різних типів файлів.

Нині фактичними стандартами є ZIP, ARJ і, мабуть, ще LZH. Також, деякі сучасні архіватори дозволяють працювати із новим кросплатформеним форматом JAR (Java ARchive), що який був створений для пересилання багатокомпонентних Java-аплетів, і водночас може застосовуватися під час роботи із упакованими архівами загального призначення (у стандарті JAR застосовуються ті ж самі методи стиснення, що і в ZIP). Маловідомий формат, CAB, що запропонований фірмою Microsoft, має засоби роботи із форматами ZIP і ARJ.

Досі росте популярність формату RAR і подібних програм завдяки високій ступені стиснення, однак стандартом він так і не стастаєла, скоріш за все, через неповороткість під час роботи із великими архівами. Так чи інакше, вдалим вирішенням задачі сумісності є створення архівів у вигляді EXE-файлів. Багато програм здатні самостійно створювати EXE-архіви на основі свого "рідного" формату. Проте, такий піхід не передбачає достатньої

гнучкості (скажімо, не дозволяє без спеціалізованих інструментів довільно «витягати» файли із архіву).

ОС Windows подарувала архіваторам графічний інтерфейс і Windows-програми більш універсальні із точки зору сумісності за форматами і до того ж використовують такі переваги ОС, як можливість давати об'єктам довгі імена і переносити довільні файли із одного додатку в інший.

Найбільш популярні із розглянутих засобів належать до категорії умовно-безкоштовних, і часто некомерційні розробки поступаються різноманіттям функцій, сумісністю і зручністю. Зауважимо, що прогалин у ефективності стиснення немає. Лідером огляду засобів архівації у ОС Windows є Zip-типізовані утиліти ZipMagic компанії Mijenix, Zip Explorer Pro фірми Aeco Systems і відома утиліта WinZip від Nico Mak Computing. Усі вони на високому рівні забезпечують сумісність із великою кількістю форматів, досить зручні у використанні. По факту, не поступається лідерам у зручності і можливостям програма WinRAR, але орієнтована на вже не дуже популярний формат RAR, не живлячись на те, що все ж забезпечує більшість ключових функцій під час маніпулювання Zip-архівами.

1.6.2 Сучасні засоби архівації даних у Mac OS X

Сучасні засоби архівації даних у Mac OS X передбачають автоматичну підтримку zip–архівів, одного із найпоширеніших форматів архівування, що підтримується в багатьох ОС. Для того, аби створити zip–архів, потрібно виділити файл в Finder'і, далі клікнути правою кнопкою мишки (або застосувати комбінацію ctrl+клік) і далі обрати в контекстному меню пункт "Стискувати". Далі створюється zip–файл із копією файлів і назвою "Архів.zip", а якщо файлів, що архівуються, більше 1, або якщо файл 1, то назва архіву співпаде із іменем початкового файлу.

Процеси розпаковування zip–файлів у Mac OS X відбуваються так само аналогічно просто, як і процеси архівація. За умовчанням "Утиліта Архівації" запускається автоматично за подвійним клацанням по zip-архіву, і це призводить до розпаковування вмісту архіву в однойменну папку, якщо файлів більше 1. Плюс, цього оригінальний zip-файл залишається у своєму

початковому вигляді, тобто під час розпаковування створюється копія вмісту архіву.

Така вбудована підтримка формату zip є зручною також під час викачуванні файлів із Інтернету засобами Safari. Коли zip-архів скачаний, його автоматично буде розпаковано вже знайомою нам “Утилітою Архівації”.

Робота с архівами з Терміналу Mac OS X передбачає наявність декількох утиліт, скажімо, таких, як gzip, bzip2, tar. Зауважимо, якщо перших 2 – це архіватори, а от tar є спеціалізованим “пакувальником” файлів. Він групує кілька файлів в один без стиснення. Застосовується tar переважно разом із gzip і bzip2, що по “Unix–традиції” можуть архівувати лише 1 файл. Тому, як це відбувається зазвичай, спочатку упаковують файли або папки із файлами в tar-архів, а потім отриманий новий файл архівують засобами gzip чи bzip2. Також, варто відзначити, що bzip2, архівує значно краще, ніж стандартний zip або gzip, хоч і робить це значно повільніше. Відзначимо також, що gzip застосовується в Mac OS для архівації логів.

У Mac OS X є вбудовані засоби розпаковування jar-архівів, причому це можна зробити кількома способами.

Перший спосіб є традиційним. Користувачу знадобиться звична програма WinRAR. Аби розпакувати JAR-файли, користувач може скористатися кількома способами, які передбачають встановлення на PC програми-архіватора WinRAR. Її легкоможна скачати із Мережі, оскільки вона знаходиться у вільному доступі.

Перший спосіб.

1) Створіть папку, куди треба буде розпакувати сам JAR-файл. Далі натисніть правою кнопкою миші на обраний JAR-файл, що необхідно розпакувати, і оберіть там функцію «Відкрити за допомогою…», а потім в списку, що з’явився програм – треба обрати опцію WinRAR.
2) Після відкриття JAR-файлу за допомого. програми WinRAR ви можете бачити список всіх об’єктів архіву. Щоб виконати розархівування, треба виділити всі файли, що з’явилися і перетягнути їх у новостворену папку.

Другий спосіб.

1) Треба натиснути правою кнопкою миші на JAR-файл і там обрати функцію «Властивості…». У вікні треба клікнути вже лівою кнопкою миші на пункт "Змінити…», а потім вибрати так само, як програму для відкриття WinRAR.
2) Поставте галочку навпроти функції під назвою «Використовувати для всіх файлів такого типу» і далі натисніть ОК. Після таких маніпуляцій, тепер можете відкривати необхідний JAR-файл і виконувати його розархівування.

Якщо вирішили використовувати другий спосіб разархівування JAR-архіву, то під час наступної ітерації процес стає ще простішим. Тепер для цього варто лиш натиснути правою кнопкою миші по необхідному Вам JAR-архіві і в функціях вибрати опцію «Витягнути файли…». Після цього на екрані ПК з'явиться вікно під назвою «Шлях і параметри вилучення…», де для процесу розархівування потрібно лише обрати цільову папку і натиснути кнопку ОК.

В якості альтернативного варіанту вирішення цієї задачі може стати процес розпакування JAR-архіву шляхом зміни його розширення. Для виконання цього натисніть послідовно кнопки «Пуск» - далі «Панель управління» - далі «Властивості папки» - далі «Вид» - далі «Приховувати розширення для зареєстрованих типів файлів» (точніше, тут треба прибрати галочку) – і тиснемо «ОК». Після цього можете змінити розширення необхідного файлу із JAR на RAR і відкрити його. Потім треба розархівувати файл будь-яким із вищеописаних способів.

Нестандартний спосіб – використати утиліту .dmg. Серед стандартного набору програм в операційній системі Mac OS є програма під назвою «Дискова Утиліта» (Disk Utility), яка початково не призначена для архівації файлів, але за її допомогою можна вионувати процеси архівування. Ідея така полягає у створенні стиснутих dmg–образів. Для цього потрібно в програмі «Дискова Утиліта» перейти за адресою "Файл–>Новий–>Образ диска із папки" і в діалоговому вікні вибрати тепапкуку із файлами. Далі, в наступному діалозі, призначеному для процесів збереження dmg-образу

потрібно вибрати ім'я і формат образу "архівний". Тестування на степінь процесів стиснення та архівації в dmg и zip приблизно одинакові – стиснуті dmg-образи стають незначно (10%–15%) більшими за розміри zip-архівів у загальному. Для прикладу розглянемо різні типи файлів разміром в 100 МБ (Таблиця 1.')

Таблиця 1.1 – Результат роботи процесів стиснення та архівації файлів

Тип файлів	Утилита, zip	Утилита, dmg
Текст (.txt) (100 МБ)	40,9 МБ	48,1 МБ (-17%)
Word (.doc) (100 МБ)	38,3 МБ	44,5 МБ (-16%)
PDF (текст) (100 МБ)	89,3 МБ	90,5 МБ (-13%)
Змішаний (txt+doc+pdf, 300 МБ)	168,5 МБ	182,4 МБ (-8%)

Далі розглянемо декілька засобів, на які варто звернути увагу, якщо стандартні засоби Mac OS X не задовільняють вимогам користувача.

Stuffit Expander. Основні його переваги – це інсталювання без оплати і уміння працювати з різними типами форматів архівів. Програмний засіб Stuffit Expander взагалі не уміє архівувати, для цього є платний Stuffit Deluxe.

Stuffit Deluxe. Вартість цього засобу архівації складає $79, він має набір опцій значно більший, ніж попередник: 1) уміє оперувати більш ніж 20-ма форматами архівів, серед яких rar, sitx, cab; 2) у випалку стиснення даних під ОС Mac, або якщо необхідно розбивати архіви на томи з метою викладання у файліві сховища, то цей програмний засіб найбільш ефективний.

The Unarchiver. Є достойною заміною Stuffit Expander, а також, вбудованому архіватору в ОС Mac. Із його влучної назви вже зрозуміло, що цей програмний засіб призначений для розпаковування файлів. The Unarchiver підтримує множини форматів архівації (включно з sitx від розробників Stuffit), успішно інтегрується із Finder, не виникає проблем із не англійськими назвами файлів. The Unarchiver є безкоштовним засобом, до того ж і open-source.

RAR для Mac OS X. Чогось такого на зразок WINRAR для ОС Mac поки що не існує. Як наслідок, користувачі rar-формату цієї ОС використовують незручну утиліту командного рядка. Поширюється цей програмний засіб безкоштовно як 40-денна версія, і якщо іноді треба архівувати файли в rar, це можна зробити із режиму Терміналу.

BetterZip. Має помірну ціну у $19,95 і є досить популярним архіватором, що одночаснопідтримує багато форматів, як то: IP, SIT, TAR, Gzip, Bzip2, RAR, 7-zip, CPIO, ARJ, Lzh/lha, JAR, WAR, CAB, ISO, CHM, RPM, DEB, NSIS, BIN, HQX, DD. Також, має підтримку швидкого попереднього перегляду архіву через утиліту Quick Look. Для цього треба викаматичати спеціальний плагін.

iArchiver. Має вартість $26. Як і Betterzip, цей архіватор написаний для для ОС Mac і підтримує досить немало багато форматів, наприклад: Zip, DMG, 7-zip, Tar, Gzip, Bzip2, Z і CPIO; вміє розпаковувати: Zip, RAR, 7-zip, Stuffit, Gzip, Bzip2, ARJ, Z, LHA, DMG, hqx, rpm тощо. До того ж, може конвертувати архіви rar в zip. В цілому його інтрефейс та юзабаліті є простими і зрозумілими.

1.7 Постановка завдань дослідження. Формулювання змісту основних завдань роботи

Завданням даного дослідження є вивчення існуючих алгоритмів, методів та засобів, що супроводжують процеси стиснення та архівації даних, а також, розробка модифікації алгоритму LZ для процесу стиснення та архівації однотипних даних та файлів.

Результатом виконання даного дослідження має стати розробка метод та кросплатформеного засобу архівації однотипних файлів. У якості об’єкту для тестування розробленого програмного засобу використаємо реляційні бази даних та їх ступінь стиснення та архівації.

Алгоритм LZ був обраний, оскільки він виконує процеси стиснення та архівації даних без втрат, є досить простим в практичній (програмній також) реалізації та актуальним нині, тому що LZ широко застосовується в інших

методах стиснення та архівації в якості засобу для архівації однотипних файлів.

У загальному, варто зазаначити, що великою перевагою методів стиснення та архівації без втрат над методами стиснення та архівації із втратами полягає в тому, що лиш вони можуть бути використані при стисненні текстових даних, оскільки ці дані мають зберігатися у первинному стані та не повинні бути спотвореними. Натомість, алгоритми із втратами під час архівації значно є кращими за ступінню стиснення, але є неефективними під час застосування до текстових даних. Ці методи часто застосовуються для процесів стиснення та архівації звуку або зображень. Тоді розпакований файл може досить потужно відрізнятися від оригіналу на рівні порівняння типу «біт у біт», проте майже не має різниці при цьому для людського вуха або ока переважно під час більшості практичних дослідів.

В результаті проведеної роботи були розглянуті сучасні методи та засоби архівації та стиснення даних, зокрема, був розглянутий алгоритм LZ77 та інші найбільш популярні алгоритми для організації процесів стиснення та архівації даних. Це такі алгоритми, як такі як LZW, алгоритм Хаффмана, PPM, RLE та інші.

Алгоритм LZ77 є родоначальником цілого сімейства словникових схем – так званих алгоритмів із ковзаючим словником, або ковзаючим вікном. Метод кодування згідно із принципом ковзаючого вікна враховує вже раніше зустрічалося інформацію, тобто інформацію, яка вже відома для кодувальника і декодувальник (друге і наступні входження деякою рядки символів в повідомленні замінюються посиланнями на її перше входження).

2 МЕТОД ТА КРОСПЛАТФОРМЕНИЙ ЗАСІБ АРХІВАЦІЇ ОДНОТИПНИХ ФАЙЛІВ

2.1 Процеси архівації за кросплатформеною технологією Java

З метою зберігання класів мови програмування Java і пов’язаних із нею ресурсів в технології Java застосовуються стиснуті архівні jar-файли [24].

Для роботи із такими стиснутими архівами в специфікації техноогії Java розроблено та представлено 2 пакети:

- java.util.zip
- java.util.jar

кожен з яких відповідно для розпакування архівів zip і jar. Відмінність даних форматів jar і zip полягає тільки в розширенні архіву zip. Стандартний технологічний пакет java.util.jar працює як аналог пакету java.util.zip, за винятком реалізації у ньому конструкторів і ключового методу void putnextentry(Zipentry e) класу Jaroutputstream. Рзглянемо показовий спеціалізований пакет java.util.jar. З метою «переробити» всі наведені там приклади на використання zip-архіву, достатньо по коду замінити «Jar» на «Zip».

Технологічний пакет java.util.jar дозволяє читати, створювати і міняти файли форматів *.jar, а також визначасти контрольні суми (КС) вхідних потоків даних. Клас Jarentry (є прямим підкласом від Zipentry) застосовується для подання доступу до записів jar-файлу. Одними з найбільш важливих методів класу є такі ключові методи [25]:

- void setMethod(int method) – метод, що встановлює спосіб стискання запису;
- int getMethod() – метод, що повертає спосіб стискання запису;
- void setComment(String comment) – метод, що встановлює коментар до процесу запису;
- String getComment() – метод, що повертає коментар до запису;
- void setSize(long size) – метод, що встановлює початковий розмір нестиснутого запису;
- long getSize() – метод, що повертає номер початкового нестисненого запису;
- long getCompressedSize() – метод, що повертає номер кінцевого стисненого запису.

В класі Jaroutputstream є така можливість запису даних в потік виведення в jar-форматі. Він перевизначає спеціальний метод write() так, щоб будь-які дані, що є в процесі запису в потік, заздалегідь стискувалися. Основними методами даного класу є такі ключові методи [26-28]:

- void setLevel(int level) – метод, що встановлює рівень стиснення. І щобільшим є рівень стиснення, тим повільніше стає робота із таким файлом;
- void putNextEntry(ZipEntry e) – метод, що записує в потік новий jar-запис. До того ж, він переписує дані із екземпляра класу Jarentry в потік виведення даних;
- void closeEntry() – метод, що завершує записування в потік jar-запису і заносить надлишкову додаткову інформацію про неї в потік виведення;
- void write(byte b[], int off, int len) – ключовий метод, що записує дані із буфера b починаючи із позиції off довжиною len в потік виведення;
- void finish() – метод, що завершує записування даних jar-файлу в потік виведення без закриття потоку.

У якості прикладу наведемо програмний код у Лістингу 2.1.

Лістинг 2.1 – Приклад застосування методів

```
void close() – закриває потік запису
/* пример # 12 : створення jar-архіву: PackJar.java */
package chapt09;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.jar.JarEntry;
import java.util.jar.JarOutputStream;
import java.util.zip.Deflater;
public class PackJar {
public static void pack(String[] filesToJar,
String jarFileName, byte[] buffer) {
try {
JarOutputStream jos =
new JarOutputStream(
new FileOutputStream(jarFileName));
// метод процесу стиснення та архівації
jos.setLevel(Deflater.DEFAULT_COMPRESSION);
for (int i = 0; i < filesToJar.length; i++) {
System.out.println(i);
jos.putNextEntry(new JarEntry(filesToJar[i]));
```

```
FileInputStream in =
new FileInputStream(filesToJar[i]);
int len;
while ((len = in.read(buffer)) > 0)
jos.write(buffer, 0, len);
jos.closeEntry();
in.close();
}
jos.close();
} catch (IllegalArgumentException e) {
e.printStackTrace();
System.err.println("Некоректний аргумент");
} catch (FileNotFoundException e) {
e.printStackTrace();
System.err.println("Файл не знайдено");
} catch (IOException e) {
e.printStackTrace();
System.err.println("Помилка доступу");
}
}
public static void main(String[] args) {
System.out.println("Створення jar-архіву");
// масив файлів для процесу стиснення та архівації
String[] filesToJar = new String[2];
filesToJar[0] = "chapt09//UseJar.java";
filesToJar[1] = "chapt09//UseJar.class";
byte[] buffer = new byte[1024];
// ім’я отриманого архіву
String jarFileName = "example.jar";
pack(filesToJar, jarFileName, buffer);
}
}
```

Оголошений клас Jarfile забезпечує досить гнучкий доступ до записів, що зберігаються в структурі jar-файлу. Це досить ефективний спосіб, бо доступ до даних та інформації у файлах здійснюється значно швидше, аніж під час зчитування кожного окремого елемента запису. Однак, відзначимо, що єдиним недоліком даного класу є те, що доступ він може здійснюватися лише для процесів читання даних. А от новий метод entries() витягує всі записи із струкутури jar-файлу. Цей метод повертає список екземплярів класу Jarentry, при чому, по одному пункту зі списку для кожного запису в структуру jar-файлі.

Розгляднемо ще метод getentry(String name), який «витягує» запис по значенні імені. Його дуг, товариш і брат, метод getinputstream() створює потік введення даних для виконання процесу записування. Цей метод повертає той

потік введення, який може використовуватися прикладною програмою для процесів читання даних із вмісту запису.

Розгянемо також клас Jarinputstream, основна функція якого читати інформацію та дані в jar-форматі із потоку введення. Він перевизначає метод read() таким спеціальним чином, щоб будь-які дані чи інформація, що прочитуються із потоку, заздалегідь розпаковувалися.

Наведемо приклад у лістингу 2.2.

Лістинг 2.2 – Приклад застосування методів

```
/* зчитування jar-архіву: UnPackJar.java */
package chapt09;
import java.io.*;
import java.util.Enumeration;
import java.util.jar.JarEntry;
import java.util.jar.JarFile;
public class UnPackJar {
private File destFile;
// размір буфера для розпакування
public final int BUFFER = 2048;
public void unpack(String destinationDirectory,
String nameJar) {
File sourceJarFile = new File(nameJar);
try {
File unzipDestinationDirectory =
new File(destinationDirectory);
// відкриття zip-архіву для зчитування
JarFile jFile = new JarFile(sourceJarFile);
Enumeration jarFileEntries = jFile.entries();
while (jarFileEntries.hasMoreElements()) {
// вилучення поточного запису із архіву
JarEntry entry =
(JarEntry) jarFileEntries.nextElement();
String entryname = entry.getName();
//entryname = entryname.substring(2);
System.out.println("Extracting: " + entry);
destFile =
new File(unzipDestinationDirectory, entryname);
// визначення каталога
File destinationParent =
destFile.getParentFile();
// створення структури каталогів
destinationParent.mkdirs();
// розпакування запису, якщо він не каталог
if (!entry.isDirectory()) {
writeFile(jFile, entry);
}
}
jFile.close();
} catch (IOException ioe) {
```

```
ioe.printStackTrace();
}
}
private void writeFile(JarFile jFile, JarEntry entry)
throws IOException {
BufferedInputStream is =
new BufferedInputStream(
jFile .getInputStream(entry));
int currentByte;
byte data[] = new byte[BUFFER];
// запис файлу на диск
BufferedOutputStream dest =
new BufferedOutputStream(
new FileOutputStream(destFile), BUFFER);
while ((currentByte = is.read(data, 0, BUFFER)) > 0){
dest.write(data, 0, currentByte);
}
dest.flush();
dest.close();
is.close();
}
public static void main(String[] args) {
System.out.println(
"Вивільнення файлу із jar-архіву");
// розташування та назва архіву
String nameJar = "c:\\work\\example.jar";
// куди розпаковувати файли
String destination = "c:\\temp\\";
new UnPackJar().unpack(destination, nameJar);
}
}
```

2.2 Використання JAR файлів в кроплатформеній технології Java

В кроплатформеній технологієї Java JAR файл – це спеціалізований java-архів (скорочення від англ. Java Archive) і є по своїй природі звичайним (як не дивно) zip-архівом, який в собі міститьдеяку частину програми, що написано мовою програмування Java [29-33].

Розвиток технології аплетів спричинив широку популярність технології Java. Використання jar-архівів водночас надає розробникам аплетів ряд таких переваг:

1) Відчутне підвищення ефективності завантаження даних з мережі. Замість декількох файлів, що створені методами окремих класів завантажується один-єдиний файл jar-архіву. І таким чином, стає зручнішим саме сховище файлів, оскільки файли класів зберігаються в єдиному стисненому файлі архіву.
2) Значне підвищення захищеності інформації та даних. У сам jar-файл можна помістити унікальний цифровий підпис, що дає кінцевому користувачеві повні гарантії, що його файл архіву не змінився із моменту її внесення. У тому випадку ж, якщо кінцевий користувач довіряє на 100% вашій фірмі, тоді він може надати наперед підписаним вами аплетам повне право доступу до його жорстких дисків або інших захищених важливих ресурсів його системи.
3) Цілковита незалежність від платформи. Виняиково із самого початку побудова jar-архівів базується на повному використанні популярної програми PKZIP для стиснення файлів. Проте jar-файли, як такі, можуть, безперечно, створюватися і зберігатися на будь-якій обчислювальній чи комп'ютерній платформі.
4) Здатність до розширюваності. До стандартної специфікації jar-файлів нині внесені серйозні доповнення, що забезпечують можливість подальших її розширень.

Важливим у підвищенні продуктивності полягає в ефективності завантаження файлів та даних. Під час роботі із протоколом прикладного

рівня HTTP (80 порт) можна виконувати процеси передавання декількох невеликих файлів. У кожному з випадків потрібно буде встановити нове транспортне з'єднання TCP/IP, яке після передавання даного файлу буде, звісно, розірвано. Встановлення кожного з'єднання на транспортному рівні пов'язане із додатковим навантаженням на сервер і мережу в цілому. Під час використання комутованих ліній зв'язку чи мереж з комутацією каналів середній час встановлення з'єднання засобами протоколу TCP/IP може складати близько 0,5 с. І це немало! Зауважимо, що для роботи одного аплета необхідно приблизно 16 файлів, підрахуємо загальний час, що витрачається лише на встановлення з'єднань - 8 с і отримаємо приголомшливі результати.

2.2.1 творення JAR-архіву

З метою створення і модифікації jar-архівів користувачам можна використовувати будь-яку програму, що успішно підтримує формат PKZIP. У загальному випадку, Jar-файл відрізняється від zip-файлу наявністю додаткового текстового файлу, так званого «файлу опису» (англ., manifest file). Цей manifest file файл містить розширені дані про всі поміщені в даний архів файли і папки. До складу опису файлу manifest file мають входити певні елементи, зокрема він має містити таке [34]:

1) Номер версії стандарту JAR. У відповідності, із цим номером версії і побудований даний JAR-архів. Цей номер версії задається спеціальним параметром Manifest-version і є обов'язковим. У середовищі SDK, скажімо, 1.2 значення цього параметра має дорівнювати 1.0
2) Мінімальний номер версії утиліти JAR, що зможе прочитати цей окремий архів. Цей параметр, по суті, є необов'язковим і має назву «Required-version».
3) Має бути окремий запис для всіх, переміщених в архів, файлів. Насправді необов'язковим є перелічувати всі поміщені в архів файли, проте досить вказати файли їх головних класів.

Вже неіснуюча фірма Sun у свій час надала для створення JAR архівів свій спеціалізований інструмент, що підходить для всіх платформ. Розглянемо

ж тепер створення jar-архіву за допомогою цієї відомої утиліти під назвою «jar».

Припустимо, що у розробника є каталог, що містить декілька програмних файлів типу .class і також містить підкаталог із стандартним ім'ям «images», що містить декілька графічних файлів типу .gif. І припустимо, що ім'я створюваного архіву буде «archive.jar».

Опишемо для користувача загальний формат типової команди виклику утиліти jar: вкажемо такі jar-параметри, як імена_файлів. Параметром для імена_файлів є цілий список імен файлів, першим іменем в якому завжди вказується ім'я самого архівного файлу. Щодо призначення ж інших імен файлів залежить, то воно цілком залежить від ключів, які можуть бути такими:

- C – ключ, що застосовується аби створити новий архів.
- N – ключ, що застосовується аби створити використовувати зовнішній файл опису, ім'я якого вказане другим в списку імена_файлів.
- M – ключ, що застосовується аби не створювати файл опису.
- T – ключ, що застосовується аби створити вивести вміст вказаного архівного файлу.
- X – ключ, що застосовується аби витягувати файли, вказані в списку «імена_файлів». Якщо імена не вказані, то витягувати абсолютновсі файли.
- F – ключ, що застосовується аби вказати, що ім'я архівного файлу є першим в списку імена_файлів.
- V – ключ, що застосовується аби вказати, що утиліта має супроводжувати повідомленнями виконання всіх дій, заданих іншими параметрами.
- 0 – ключ, що застосовується аби збереження файлів в архіві виконувалося без їх стиснення.
- U – ключ, що вказує, що потрібно відновити деякі вказані файли. Або у випадку застосування команди «jar umf manifest імя_архіву» вказує що потрібно відновити інформацію у файлі опису.

– I – ключ, що вказує, що необхідно згенерувати файл INDEX.LIST, який містить інформацію про всі файли архіву.

Отже, для створення нового архіву в командній стрічці введіть таке: jar cf archive.jar .class images/.gif

2.2.2 Додавання в jar-архів цифрового підпису

З метою розібратися, як це працює і яким чином цифровий підпис подається в jar-архіві, необхідно заздалегідь знати ключові поняття із галузі криптографії – це процес шифрування за допомогою застосування відкритого ключа. Щоб мати можливість внесення цифрового підпису до потрібного jar-архіву необхідно мати 2 таких інструменти [35-38]:

1) Спеціалізована тиліта keytool. Застосовується з метою генерації пари відкритого і закритого ключів, а також, самого сертифікату.
2) Спеціалізована утиліта jarsigner. Застосовується з метою безпосереднього приміщення цифрового підпису в jar-архів із застосуванням вже наявного у Вас сертифікату.

Ось що знадобиться для здійснення процесу додавання фифрового підпису:

1) Виконати процес генерації пари ключів (відкритого і закритого).
2) Процес отримання сертифікат на цю пару ключів - відкритого і закритого.
3) Процес використання сертифікату для прилаштування цифрового підпису в jar-архів.

Процес генерації пари ключів. Для генерації нової пари ключів (відкритого і закритого) можна скористатися такою спеціальною командою: *keytool -genkey -alias testkey.* І тоді у результаті виконання цієї команди буде створена нова пара ключів (відкритого і закритого), потім – збережена в БД під іменем testkey (рисунок 3.1).

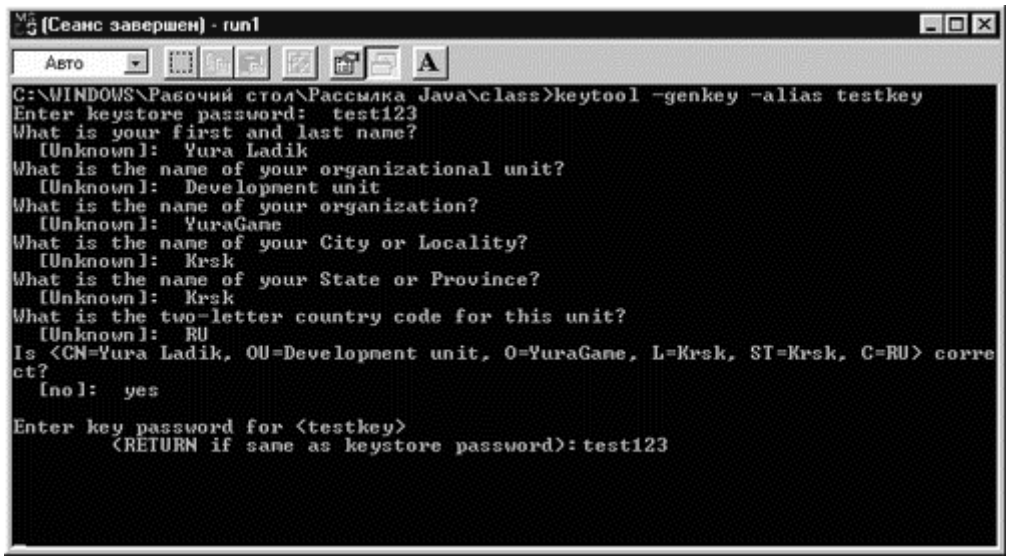


Рисунок 2.1 – Створення пари ключів

Програмний засіб під назвою keytool дозволяє вказувати такі необов'язкові параметри:

- ключ v – показує запит на виведення повідомлень про дію програми.
- ключ alias псевдонім – це є псевдонім (або ім'я) яке привласнюється цій парі.
- ключ keyalg алгоритм ключа.
  - алгоритм шифрування вашого цифрового підпису – зазвичай за умовчанням це алгоритми Sha1 із DSA і його не обов'язково вказувати якщо не будете змінювати сам алгоритм, при цьому розмір ключа при генерації DSA пари ключів може бути від 512 до 1024 бітів;
  - якщо ж ви хочете використати Md5 із RSA, тоді вкажіть таку опцію «-keyalg "RSA"». Ця опція -keyalg обумовляє також і опцію «-sigalg», а це – такий алгоритм підпису, що буде використаний за умовчанням для підписання jar-файлу (під час створення дайджесту повідомлень).
- keysize «довжина_ключа» - це ключ для визначення розміру ключів, що генеруються, вимірюється в бітах.
- keypass пароль – це ключ для задання пароля для даного ключа. Якщо пароль не буде вказаний тут, в командному рядку, тоді програма запропонує ввести його значення в режимі діалогового

вікна однозначно. До речі, довжина пароля має бути не менше 6 символів.

- keystore сховище – це ключ для визначення розташування сховища ключів.
- storepass пароль – це ключ для визначення паролю доступу до сховища ключів.
- validity valdays – це ключ, що визначатиме термін придатності вашого сертифікату. За умовчанням термін становить 180 днів, проте, можна вказати більше або менше днів.

За умовчанням утиліта «keytool» передає відкритий ключ в підписаний вами сертифікат X.509.v1, а зручною командою keytool -list можна переглянути вміст файлу keystore (рисунок 2.2)

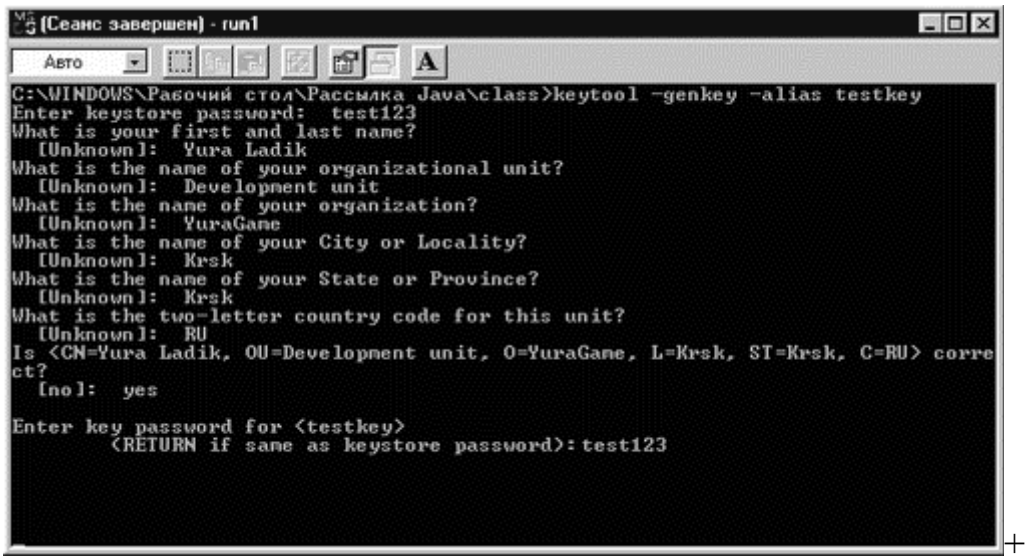


Рисунок 2.2 – Вміст «keystore»

Процес отримання сертифікату. Після виконання процесу генерації нової пари ключів необхідно виконати процес генерації запиту CSR (англ., Certificate Signing Request). Цей запит CSR надсилається в будь-яку вибрану Вами службу сертифікації. З метою генерації CSR-запиту радимо ввести таку команду: *keytool -certreq*. ДлУ нашому випадку, для нашої пари ключів «testkey» команда виглядатиме так: *keytool -certreq -alias testkey* (рисунок 2.3).

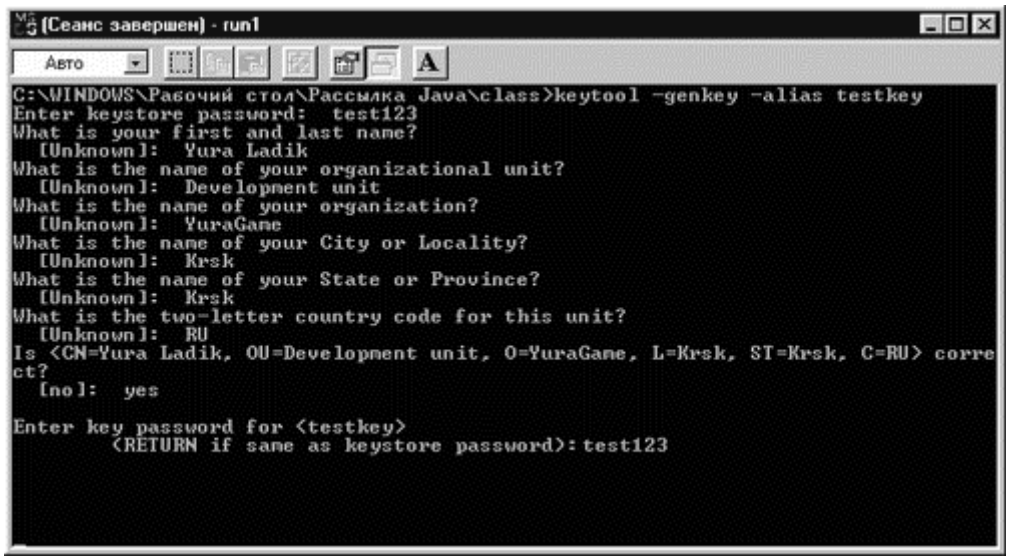


Рисунок 2.3 – Значення «testkey» для двох ключів

Під час генерації CSR запиту допускаються такі параметри [39-40]:

- v – параметр, що забезпечує супровід роботи програми видаванням повідомлень.
- alias псевдонім – параметр, що забезпечує визначення псевдоніма пари ключів, для яких необхідно отримати надійний цифровий сертифікат. За умовчанням часто застосовується значення mekey.
- sigalg алгоритм_сигнатури – параметр, що забезпечує визначення алгоритму внесення цифрового підпису.
- file csr_файл – парвизначаєаметр, що забезпечує ім’я і розташування файлу запиту, що генерується.
- keypass пароль – параметр, що забезпечує завдавання паролю для доступу до даного ключа.
- storepass пароль – параметр, що забезпечує пароль доступу до сховища ключів.
- keystore сховище – параметр, що забезпечує ім’я і розташування файлу із парою ключів (відкритим та закритим).

Є ще спеціальний Csr-запит, який після генерації відсилається у обрану Вами службу сертифікації. Лше після проведення всіх необхідних перевірок і однозначного посвідчення вашої особи, Вам буде виданий необхідний цифровий сертифікат.

У будь-якому випадку, Ви самостійно можете занести отриманий цифровий сертифікат у файл і ввести таку команду: *keytool -import*. Ці параметри можуть бути вказані в режимі import:

- v – параметр, що забезпечує супровід роботи програми процесами видачі повідомлень.
- alias псевдонім – параметр, що забезпечує наявність псевдоніму повного імені, яке використовуватиметься із даним цифровим сертифікатом.
- file файлу-сертифікату – параметр, що забезпечує ім'я і шлях до розташування файлу, де збережений отриманий цифровий сертифікат.

Команда keytool *-export -alias testkey -file* дасть команду спеціалізованій утиліті скопіювати Ваш цифровий сертифікат у вказаний файл. Далі відправте свій цифровий сертифікат всім адресатам та одержувачам, які використовуватимуть підписані Вами jar-архіви. Тепер прилаштування в jar-архів надійного цифрового підпису за допомогою утиліти jarsigner робить можливим безпечний обмін даними. Окрім прилаштування цифрового підпису в jar-архів, утиліта jarsigner здатна також додатково перевіряти на цілісність підписані jar-архіви. З цією метою достатьо запустити її із параметром -verify/.

Під час введення такої команди: *jarsigner myjarfile.jar* утилита обов'язково використає сховище ключів за умовчанням і результат буде записаний у файл myjarfile.jar, що замістить початковий вихідний файл архіву. З метою створення jar-архіву підписаного цими цифровими ключами testkey введіть таку команду: *jarsigner myJarFile.jar testkey*.

Якщо потрібно, щоб файл myjarfile.jar залишився незміненим а результат роботи ключа записався, скажімо, у файл mysignedjarfile.jar введіть ось таку команду*: jarsigner -signedjar mysignedjarfile.jar myjarfile.jar testkey.*

Можемо константувати те, що робота із програмою-архіватором вимагає роботи із командною стрічкою також. Із метою спростити роботу та дії користувача і був розроблений програмний засіб із графічним інтерфейсом в межах цієї дипломної роботи.

2.3 Метод архівації однотипних файлів

Розроблений метод призначений для процесів оптимізованої архівації великих кількостей маленьких однотипних файлів. В основі роботи запропонованого методу архівації однотипних файлів є ідея заміни повторного входження цілого блока даних посиланням на попередню позицію його входження.

2.3.1 Розробка методу архівації однотипних файлів

Загальна схема складових та процесів методу архівації однотипних файлів зображена на рисунку 2.4.

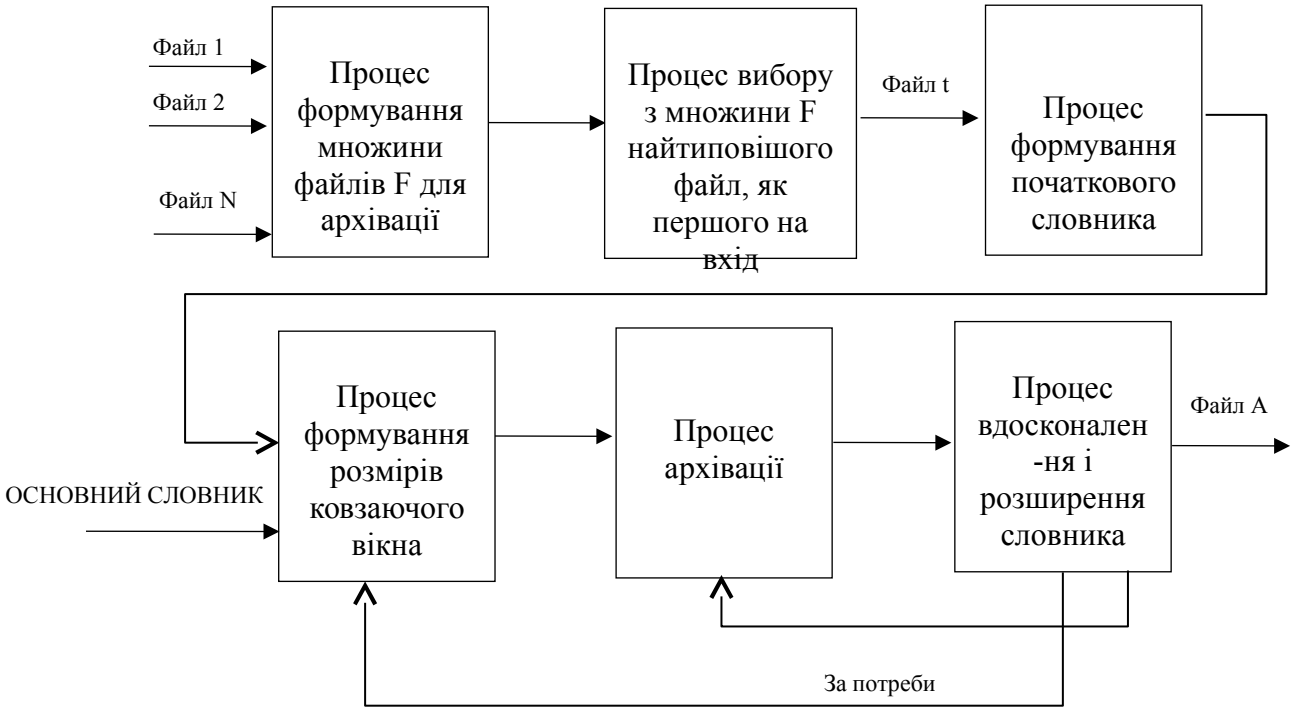


Рисунок 2.4 – Загальна схема методу архівації однотипних файлів

Метод архівації однотипних файлів передбачає забезпечення та виконання таких процесів:

1) Прийом на вхід множини файлів;
2) Процес формування множини файлів F для архівації;

3) Процес вибору з множини F найтиповішого файл, як першого на вхід;
4) Виокремлення такого ключового файлу;
5) Процес формування початкового словника;
6) Процес формування розмірів ковзаючого вікна. «Ковзаюче вікно» в даному випадку є динамічною структурою даних, що організована таким чином, аби містити в собі введену раніше інформацію та надавати до неї доступ;
7) Процес архівації. Передбачає звертання до елементів «ковзаючого вікна» і замість значень послідовності, що архівується, вставлення посилань на ці значення своєрідному «словнику»;
8) За потреби – перехід до процесу формування розмірів ковзаючого вікна.
9) Процес вдосконалення і розширення словника. Перехід на процес архівації знову;
10) Архівний файл А.

2.3.2 Розробка процесу формування початкового словника

Процес формування словника у стандартному використанні алгоритму LZ77 це формування комбінації, що повторюється і кодуються парою:

- довжина збігу (match length)
- зсув (offset) або дистанція (distance).

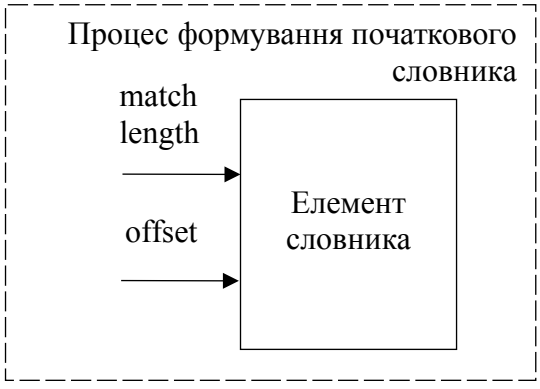


Рисунок 2.5 – Процес формування словника

Під час процесу формування початкового словника кожна така комбінація «довжина збігу + зсув» трактується як команда копіювання символів із певної позиції «ковзаючого вікна», або дослівно звучить так:

«Повернутися назад в словнику на значення «зсуву» символів і скопіювати значення «довжини збігу» символів, починаючи із поточної позиції».

2.3.3 Розробка процесу архівації

Особливість розглянутого в даному методі архівації однотипних файлів є сам процесу архівації, якій полягає в тому, що використання комбінації кодової пари «довжина-зміщення» є не тільки прийнятним, але й ефективним у тих випадках, коли значення довжини збігу *match length* перевищує значення зсуву *offset*.

В кінцевому підсумку, ступінь ефективності процесу та архівації в даному методі архівації однотипних файлів залежить від того, наскільки багато в файлі повторюваних комбінацій, і наскільки вони великі.

Це приводить нас до такої запропонованої ідеї: з метою збільшити кількість таких блоків, необхідно «об'єднати» всі файли, які планується архівувати, в один великий файл, і вже після цього його, цей великий файл, стискати. Звісно, такий підхід буде дуже ефективним для однотипних файлів (і виправдовує мету даної роботи), і вкрай неефективним для різних типів файлів, іншими словами, із різним профілем даних. Тому застосування запропонованого методу архівації однотипних файлів може давати високі результати роботи процесу архівації.

Оскільки запропонований метод тісно пов'язаний із змістом даних, що архівуються, можна говорити про ентропію.

В загальному випадку, дані, що отримуються приймачем несуть корисну інформацію, якщо має місце невизначеність відносно стану джерела даних. Величина, що визначає невизначеність окремого і-го повідомлення називають частковою ентропією (2.1).

$$M(x_i) = \frac{log_2 y}{p(x_i)} \tag{2.1}$$

І тоді кількість інформації і невизначеність для цієї множини випадкових повідомлень можна отримати шляхом усереднення по всіх елементам даних (2.2)-(2.3).

$$\mathrm{I}(x) = -\sum p(x_i) * \frac{log_a 1}{p(x_i)}; \tag{2.2}$$

$$H(x_i) = -\sum P(x_i) log_a p(x_i) \tag{2.3}$$

де $H(x_i)$ – ентропія.

Не дивлячись на те, що є співпадіння залежностей, все ж ентропія і кількість інформації є принципово різними. Сама ентропія, що виражає «середню невизначеність джерела» даних є характеристикою джерела даинх і, якщо нам доступна статистика повідомлень, яка може бути визначена заздалегідь. *I(x)* є попередньою характеристикою і визначає кількість даних, що отримуються із вхідного повідомлення. *H(x)* – це, так звана, міра нестачі інформації про стан окремої частини якоїсь системи чи системи. Пропорційно надходженню даних про стан системи, ентропія останньої стає меншою. Співпадіння виразів (2.2) і (2.3) говорить про те, що кількість отриманих даних в числововму еквіваленті дорівнює ентропії, що існує залежно від джерела даних на розглянутій частині каналу зв'язку тією кількості інформації з ентропією, що чітко проявляється діалектичний закон.

Середнє значення ентропії даних при однаковій кількості елементів може відрізнятися залежно від статистики і характеристик самих даних. Під час наявності зв'язків залежності між елементами даних, ентропія стає меншою. У випадках, коли зв'язки залежності охоплюють 2 або 3 елементи, тоді формула для обчислення ентропії стане (2.4) для 2 елементів і (2.5) для 3 елементів.

$$H(x) = -\sum\sum p\left(x_i, x_j\right) log_a p(x_i, x_j), \tag{2.4}$$

$$H(x) = -\sum\sum\sum p(x_i, x_j, x_k) log_a p(x_i, x_j, x_k) \quad (2.5)$$

Ті початкові дані є кращими, у яких показники ентропії є оптимальними. Виміром наскільки дані за своєю ентропією відрізняється оптимальних даних, покаже коефіцієнт архівації (2.6):

$$\mu = \frac{H(x)}{H(x)_{max}} \quad (2.6)$$

де $H(x)$ – реальне; $H(x)_{max}$ – ентропія відповідному йому оптимального повідомлення.

Якщо дані, що не є оптимальними і оптимальні деякі дані характеризуються однаковим значенням ентропієї, тоді справедлива така рівність (2.7)

$$n * H(x) = n' * H(x)\max, \quad (2.7)$$

де n – число елементів не оптимального повідомлення даних; n' – число елементів відносно оптимального повідомлення даних.

Оскільки ентропія для оптимальних даних є максимальною, тоді число елементів неоптимальних даних n завжди буде більшою від кількості елементів відповідних оптимальних даних n'. І тоді коефіцієнт ахівування можна виразити через кількість елементів повідомлення (2.8)

$$\mu = \frac{n'}{n}. \quad (2.8)$$

Так, реальні дані тоді при однаковій ступені інформативності володіють визначеною надлишковістю в своїх елементах у порівнянні з оптимальними даними. Дійсним коефіцієнтом архівування вважатимемо (2.9)

$$K_A = \frac{N_{in}}{N_{out.}} \quad (2.9)$$

де N_{in} – кількість двійкових розрядів на вході методу архівації. N_{out} – кількість двійкових розрядів на виході методу архівації.

В рамках даної роботи це було експериментально досліджено автором, і виявлено, що текстові файли були заархівовані гірше за файли реляційної бази даних. Саме цей факт пояснює тим, що в текстових (чи інших складних форматів) файлах, наявне досить незначне число повторюваних блоків в цих файлах.

В таблиці 2.1 наведено експериментальні дані архівації однотипних файлів по одному та однотипних файлів у кількості 10 шт. та 100 шт.

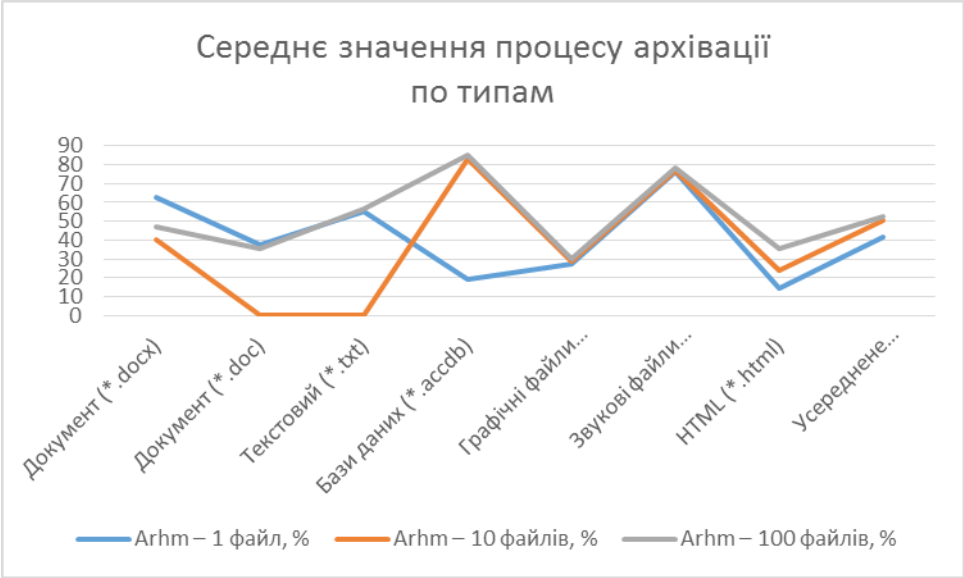
Середнє значення процесу архівації для різних типів файлів Arh_m розраховане для процесу архівації для різних типів файлів, кількість – 1 файл, процесу архівації для однотипних файлів, кількість – 10 файлів та процесу архівації для однотипних файлів, кількість – 100 файлів.

Таблиця 2.1 – Середнє значення процесу архівації

Тип файлів	Arh_m – 1 файл, %	Arh_m – 10 файлів, %	Arh_m – 100 файлів, %
Документ (*.docx)	62,875	40,33	47,4
Документ (*.doc)	37,7825	32, 567	35,7
Текстовий (*.txt)	55,305	47, 376	56,78
Бази даних (*.accdb)	18,89	83,02	85,189
Графічні файли (*.jpg)	27,31	28,5	30,01
Звукові файли (*.mp3)	76,4	77,08	78,5
HTML (*.html)	14,74	24,129	35,67
Усереднене значення показника архівування	41,9003571	50,612	52,75

На рисунку 2.6 наведено графічне представлення результатів дослідження процесів архівації і виявлено, що розроблений метод архівації

однотипних файлів гарно працює в умовах наявності не менше 10 файлів на вході методу, а при наявності більшої кількості (100 і більше) однотипних файлів значення Arh_m росте.



а)

Рисунок 2.6 – Діаграма процесу архівації

Запропонований метод архівації однотипних файлів дає приріст середнього значення процесу архівації на 10,85%. А зі збільшенням кількості однотипних файлів для процесу архівації цей показник може все більше зростати.

2.3.4 Розробка процесу формування основного словника

Незважаючи на отримані результати, навіть в таких успішних випадках результат роботи процесу архівації буде не дуже високим. Це пояснюється тим, що досить рідко трапляються повністю ідентичні блоки. Натомість, частіше, у БД, а особливо, у реляційних БД, трапляються так звані «подібні блоки», які відрізняються, умовно кажучи, кількома символами. З метою вирішення такої задачі можна використати інший підхід, заснований на принципі «ковзаючого вікна» до самого словника.

Нехай існує початковий словник, який містить уже проглянуті блоки (див. схему методу архівації однотипних файлів). Нехай приходить нова порція даних, тоді будемо зчитувати ці дані до тих пір, поки вони не співпадуть із одним із значень із початкового словника, якимось словниковим блоком звідти. Як тільки виникає така ситуація, яка «порушує» хід речей, ми включаємо до словника цей новий фрагмент також як кортеж, але тепер вже ось такого вигляду:

- код словникового блоку, який дав максимальний збіг – max match;
- довжина максимального збігу m_len;
- символ, що порушив співпадіння symbol;
- сам символ symb_m.

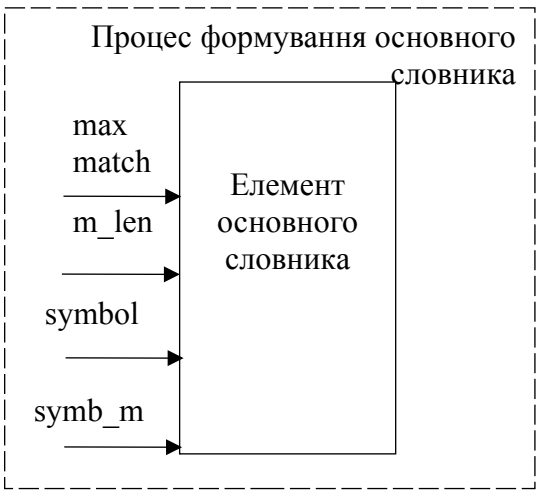


Рисунок 2.7 – Процес формування основного словника

Таким чином, основний словник розростається, проте у випадках достатньо великого числа блоків із справді незначними відмінностями, ступінь архівації дійсно зростає. Додатковий бонус – це те, що тепер уже необов’язково «зклеювати» всі вхідні файли в один великий, і тепер можна оперативно додавати чи видаляти файли з архіву, а ще можна просто шляхом додавання нових файлів коригувати основний словник.

Саме таким методом і працює процес формування основного словника, призначений для методу архівації однотипних файлів за умови наявності досить великої кількості цих однотипних файлів.

Розглянемо алгоритм роботи програмного засобу. Що реалізовує метод архівації однотипних файлів.

2.4 Розробка алгоритму роботи кросплатформеного засобу архівації однотипних файлів

Загальний алгоритм роботи кросплатформеної програми архівації однотипних файлів є таким:

1) Запуск програми.
2) Якщо задана команда виходу, перейти до пункту 19. Інакше перейти до пункту 3.
3) Виконати архівацію. Якщо початковий файл вже завантажено, йти до пункту 14. Інакше прямуй на пункт 4.
4) Вибрати ім’я нового файлу.
5) Отримати команду користувача. Якщо потрібно додати файл в список, йди до пункту 6. Якщо потрібно змінити файл, йди до пункту 7.
6) Додати файл в список. Якщо потрібно видалити файл, йти до пункту 8. Якщо потрібно створити архів, йти до пункту 9.
7) Змінити файл.
8) Видалити файл.
9) Створити архів.
10) Вибрати файл для архіву.
11) Стиснути файл та долучити до потоку.
12) Якщо архів пустий, йти на пункт 10. Інакше йти на пункт 13.
13) Зберегти потік як архів. Перейти на пункт 2.
14) Завантажити початковий файл.

15) Вибрати файл.
16) Вилучити файл із потоку і розпакувати.
17) Зберегти файл.
18) Якщо список пустий, йти на пункт 2. Інакше пункт 15.
19) Кінець роботи програми.

Загальний алгоритм роботи програми наведено на рисунку 2.8.

Далі розглянемо сам реалізацію алгоритму формування основного словника однотипних файлів, описання якого було наведено вище, це необхідно для того, щоб продемонструвати його реалізацію стандартними компонентами кросплатформеної технології Java. Потім розглянемо рішення для розробки користувацького інтерфейсу.

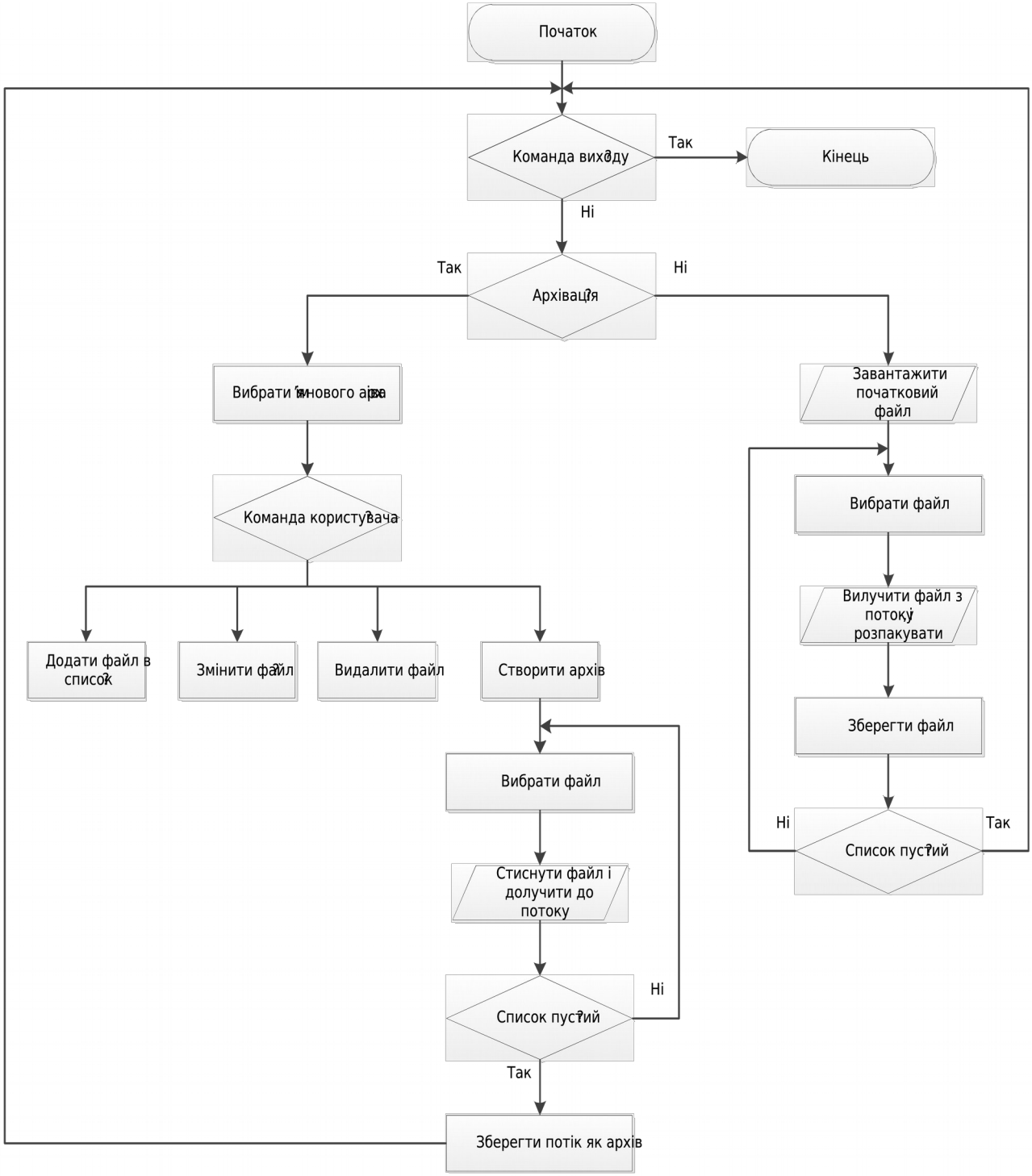


Рисунок 2.8 – Блок-схема алгоритму роботи кросплатформеної програми архівації однотипних файлів

2.5 Реалізація алгоритму формування основного словника однотипних файлів

Розглянемо блок-схему формування основного словника блоків, що зображений на рисунку 2.9.

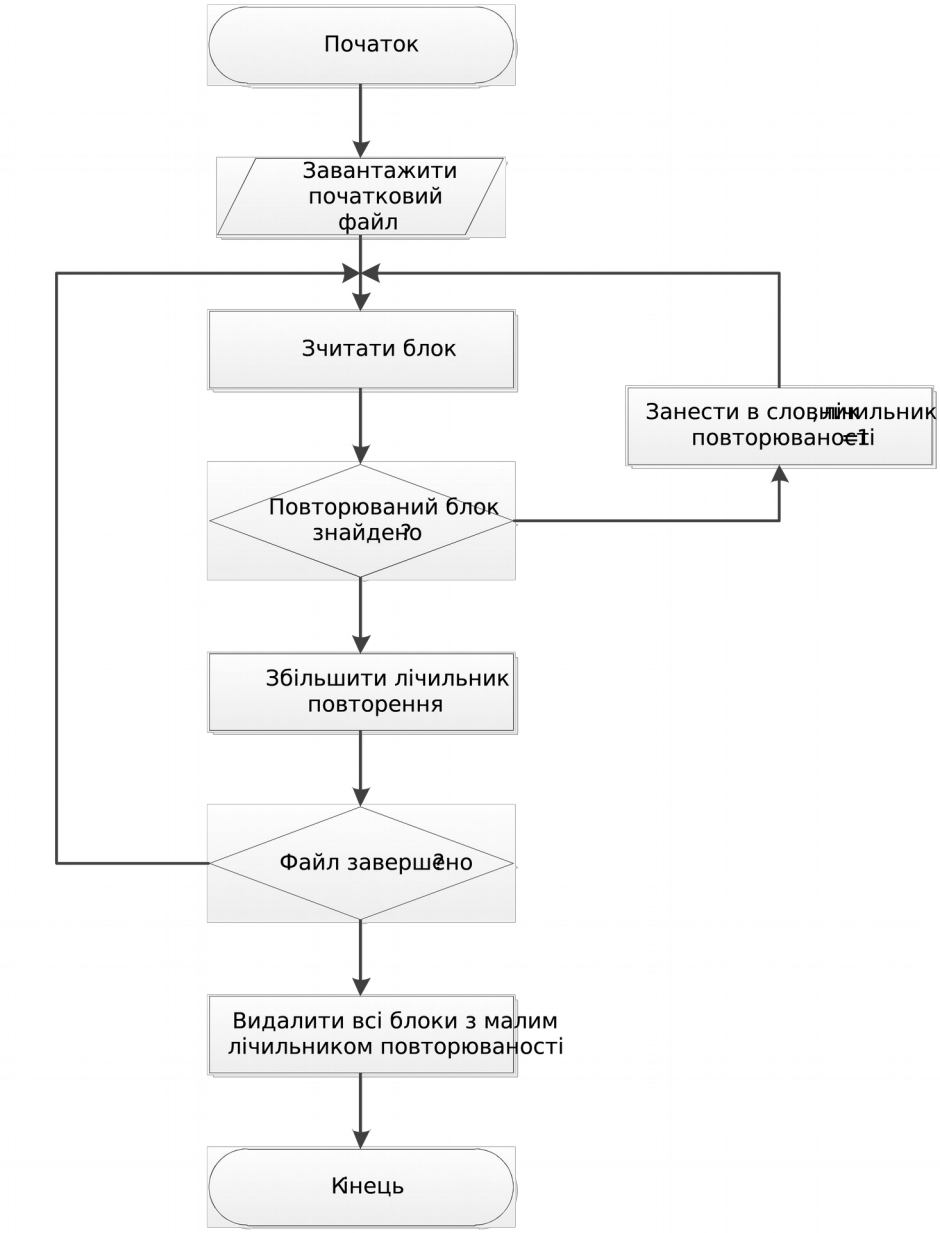


Рисунок 2.9 – Алгоритм створення основного словника

Наведений вище алгоритм створення основного словника створює основу для процесу архівації, аналізуючи перший наявний в потоці файл, шляхом виявлення в ньому повторюваних блоків. Всі наступні після нього

файли тепер є джерелами коригувань словника. Ось чому для покращення ступеню архівації за методом архівації однотипниxx файлів варто і доцільно та обгрунтовано автором в першу сергу у вхідний потік надавати найбільший чи найтиповіший файл.

Цим обумовлено вибір в технології Java класу архівного файлу JarFile, що являє собою вектор класів типу JarEntry, кожен із яких є динамічним фрагментом потоку архівації, а якщо розглядати його на нижньому рівні, то він є «обгорткою» для класу JarOutputStream. Для розуміння, всі ці класи успадковані від класів для роботи із алгоритмом Zip, який є наріжним каменем Compression API в Java [41-43].

Покажемо рисунок 2.10, де видно відношення між ZipFile та ZipOutputStream – вони точнісінько такі самі для JarFile та JarOutputStream і цілком підходять для вирішення нашої задачі [44-45].

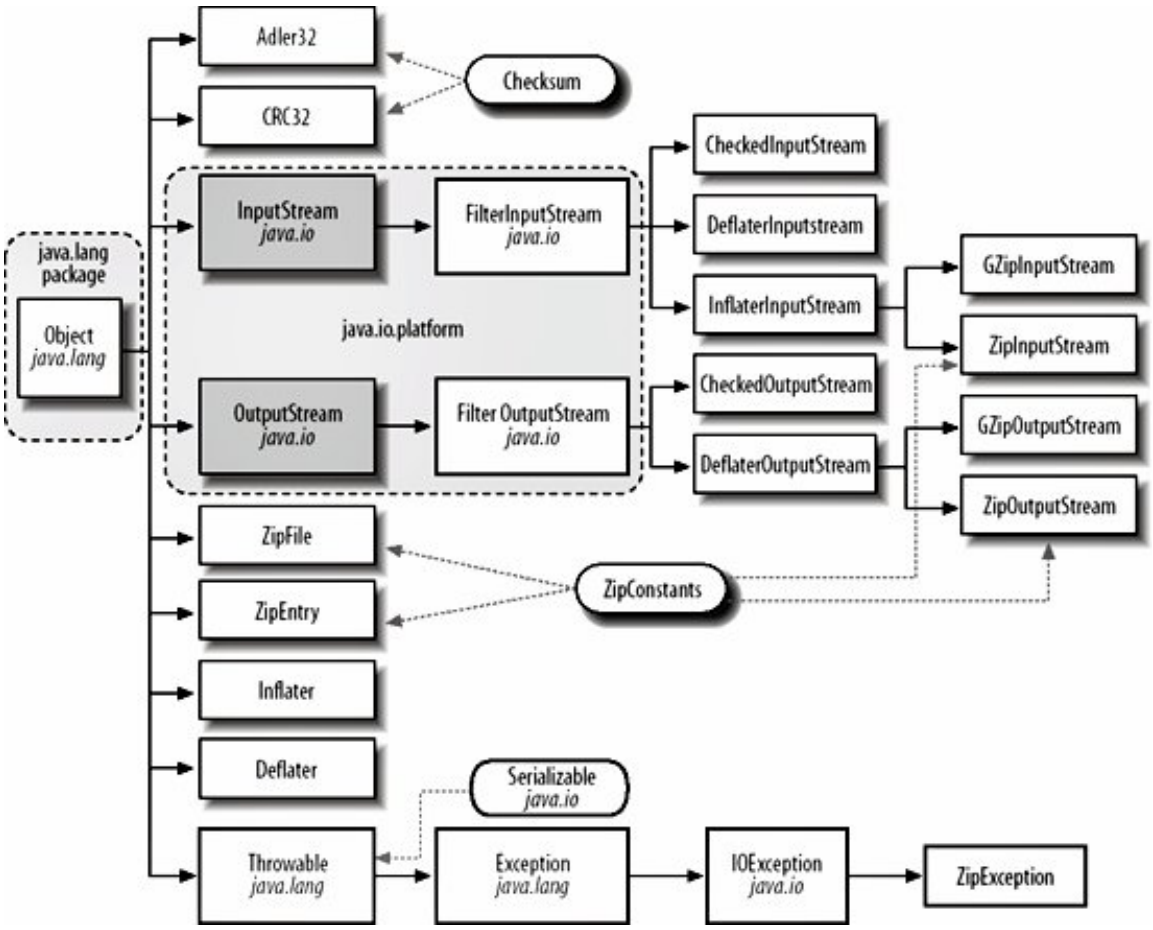


Рисунок 2.10 – Compression API

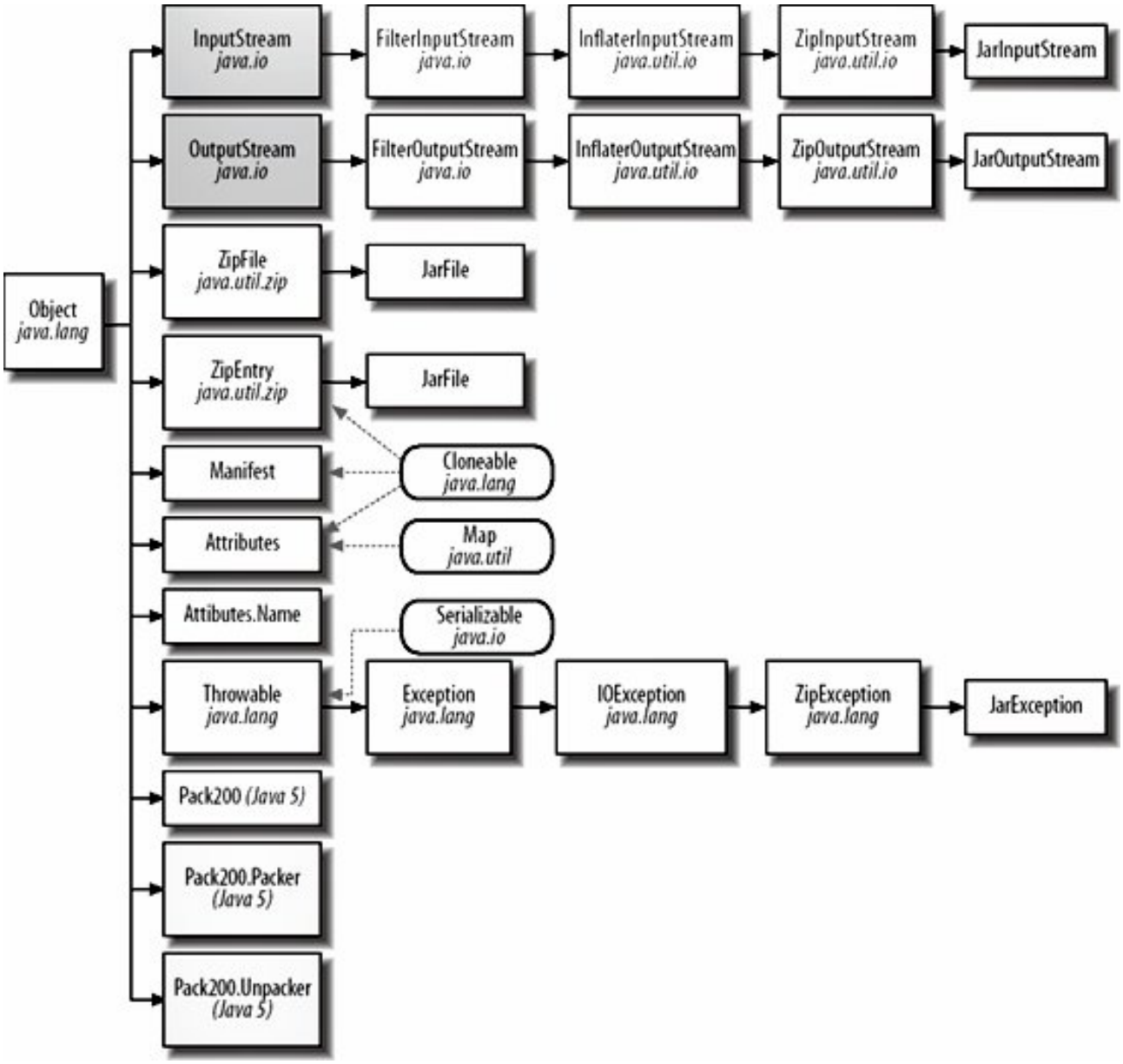


Рисунок 2.11 – Спадкування класів роботи із Jar-архівації

Напишемо код, що реалізовує процес долучення файлу до потоку архівації (див. лістинг 3.3).

Лістинг 3.3 – Долучення файлу до потоку

```
JarEntry entry = new JarEntry(current.getName());
jar.putNextEntry(entry);
jar.write(getBytesFromFile(current.getCanonicalPath()));
jar.closeEntry();
```

Зверніть увагу, що у самому потоці з самого початку відкривається вікно для стиснутого файлу JarEntry, назва якого має відповідати назві файлу,

і тоді і тільки тоді файл перетворюється на масив байтів (за замовчуванням він є одним великим блоком даних, який далі буде аналізуватись по основному словнику), і потім пересилається в потік. Розпаковування ж відбувається інакше (див. лістинг 2.4).

Лістинг 2.4 – Реалізація процесу розпаковування файлу

```
JarEntry entry = (JarEntry) entries.nextElement();
if(entry.getName().equals(name))
{
  FileOutputStream fos = new FileOutputStream(unpackPath.getText() +
File.separator + entry.getName());
  InputStream is = jar.getInputStream(entry);
  while(is.available() > 0)
   fos.write(is.read());
  fos.close();
  is.close();
     }
```

На початку реалізації даного процесу створюється дочірній потік, і із потоку стиснутого файлу послідовно зчитуються дані (байти) «на льоту», одночасно пр цьому обмінюючись даними із словника (цей фрагмент коду виділено).

Таким чином, користуючись такою можливістю в мові Java можна легко створити архіватор, орієнтований на однотипні файли.

2.6 Побудова користувацького інтерфейса для кросплатформеного засобу архівації однотипних файлів

Пакет Swing – один із найбільш комплексних компонентів та потужних інструментів мови програмування Java. Він містить повний набір елементів керування для створення та візуального конструювання графічної програми будь-якого призначення. На рисунку 2.12 показано загальну структуру цього пакету [46-47].

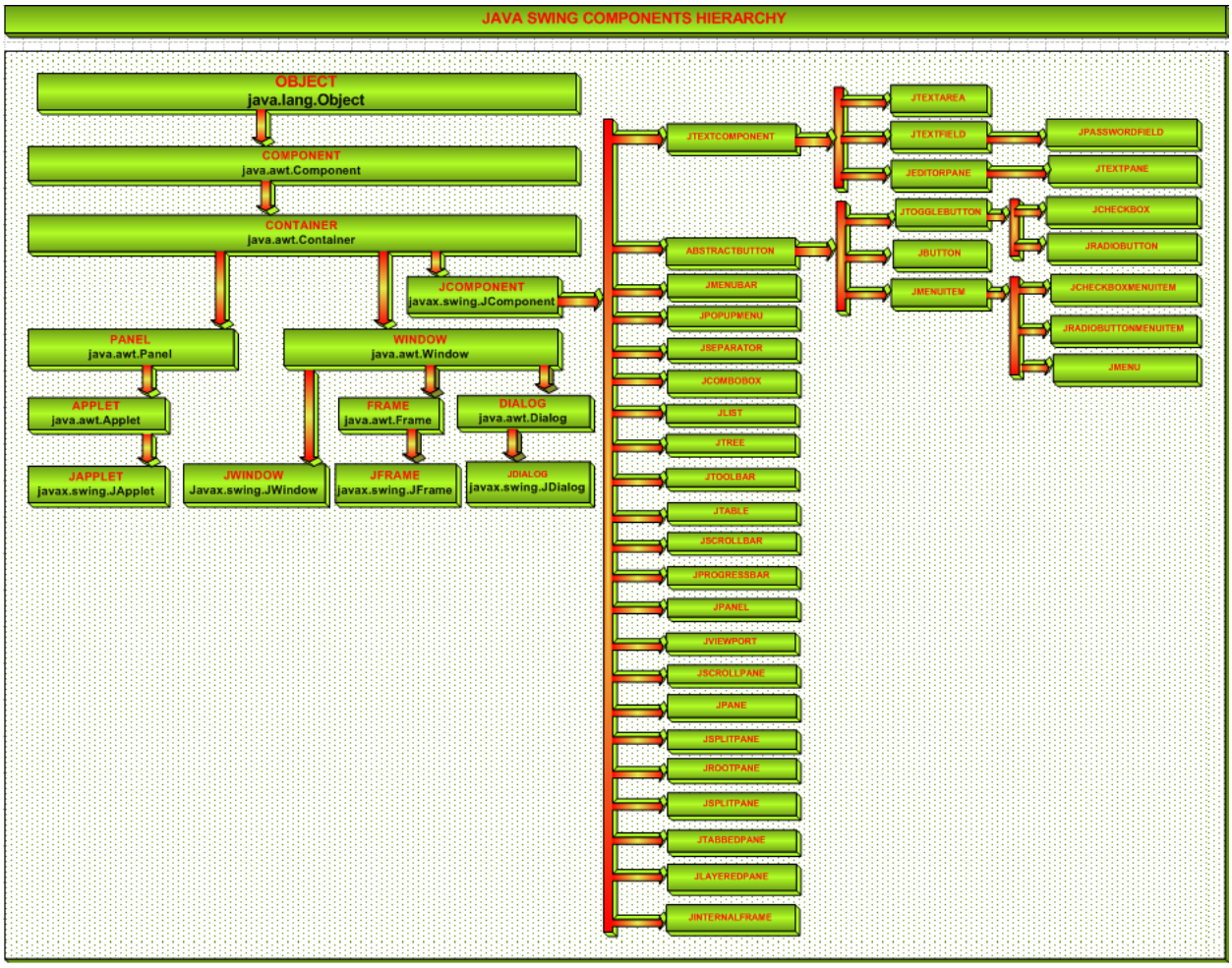


Рисунок 2.12 – Структура пакета Swing

Для реалізації поставленої в цій роботі задачі створення інтерфейсу для програми архіації однотипних файлів, знадобляться компоненти JTextField, JRadioButton, JButton та JTable. Оскільки іх створення в сучасних середовищах інтегрованої розробки автоматизоване, то додаткового опису його не будемо наводити, а зосередимося в основному на декількох особливих рішеннях.

Так, скажімо, аби вказати шлях для розпакування файлу, необхідно вибирати каталоги для цього, а не окремі файли. Отож, ми скористаємось стандартним компонентом JFileChooser, і запишемо це так (лістинг 2.5):

Лістинг 2.5 – Задання каталога для розпаковування

```
JFileChooser chooser = new JFileChooser();
chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
if(chooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION)
{
```

```
try
{
 unpackPath.setText(chooser.getSelectedFile().getCanonicalPath());
}
catch (IOException exc)
{
 System.err.println(exc.getMessage());
 exc.printStackTrace(System.err);
}
    }
```

Зверніть увагу, що для реалізації розробнику необхідно обробляти виключення IOException. Це пов’язано із тим, що в різних типах ОС записи про каталоги організовані по-різному. А програмний засіб – кросплатформений, давайте не будемо забувати.

Важливо, що, з метою коректного відображення відсотку готовності процесу архівації однотипних файлів нам знадобиться перетворити дріб подвійної точності у значно сприйнятливіший формат. Отож, скористаємось для цього особливим службовим класом DecimalFormat, і задамо так (лістинг 3.6):

Лістинг 2.6 – Відображення відсотка готовності

```
        long fullSize = Long.parseLong(fileList.getModel().getValueAt(i, 1).toString());
        DecimalFormat df = new DecimalFormat("####0.00");
        double percent = ((double)entry.getCompressedSize() / (double)fullSize) * 100;
            ((DefaultTableModel)fileList.getModel()).setValueAt(df.format(percent) + "%", i, 2);
```

Вираз в дужках «"####0.00» називається «регулярним виразом». Це є мова завдання шаблонів. У даному випадку ми задаємо 4-значне поле для частки до коми, і 2 – для частки після коми. 4-значне поле необхідне для коректного вирівнювання і кращої читабельності числа. Нарешті, використаємо радіокнопки для того, щоб в залежності від режима відкривати вікно збереження чи відкриття архівного файлу (лістинг 2.7).

Лістинг 2.7 – Варіабельне вікно вибору файлу

```
JFileChooser chooser = new JFileChooser();
chooser.setFileFilter(new JarFileFilter());
boolean accepted = false;
if(flagPack.isSelected())
{
 if(chooser.showSaveDialog(this) == JFileChooser.APPROVE_OPTION)
  accepted = true;
}
else
{
 if(chooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION)
  accepted = true;
}
```

Отже, в підсумку, побудова користувацького інтерфейсу для програми архівації однотипних файлів не є складною задачею, у випадку правильного і повного використовувати стандартних компонентів мови програмування Java.

2.7 Етапи створення програми архівації однотипних файлів

Створення програми архівації однотипних файлів мовою програмування Java поділяється на 3 основних етапи.

1) Розробка головного класу програми архівації однотипних файлів.
2) Розробка користувацького інтерфейсу для програми.
3) Реалізація основних функцій програми.

Головний клас програми архівації однотипних файлів відрізняється від інших класів наявністю так званої «головної функції», що одночасно є точкою входу в програму і виконується першою під час запуску програми.

Використаємо для створення програми архівації однотипних файлів інтегроване середовище розробки JDeveloper (рис. 3.13).

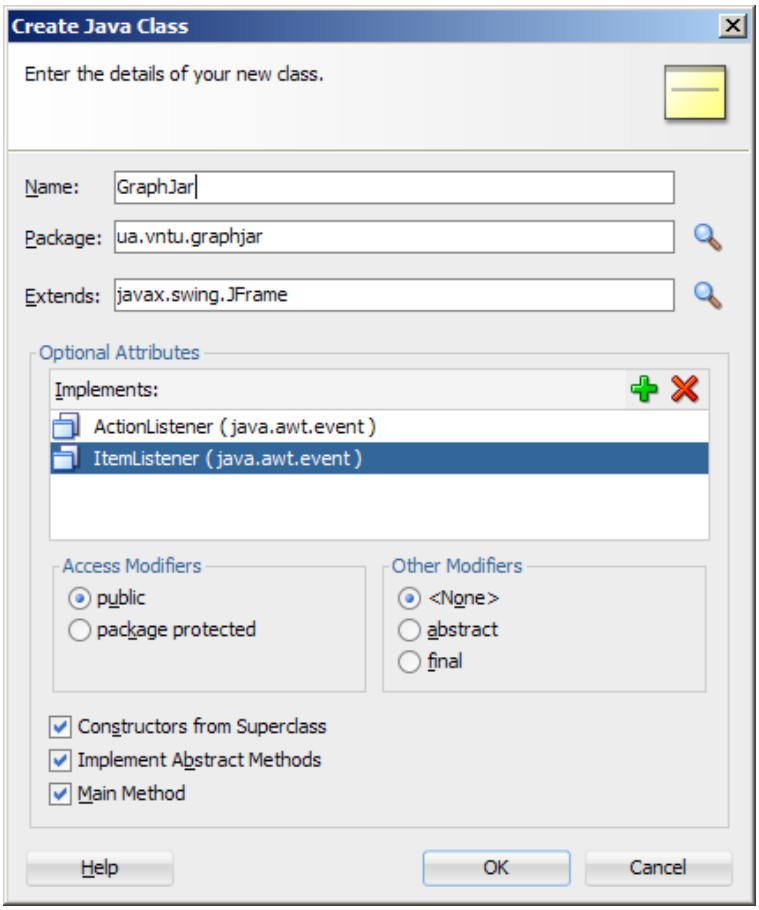


Рисунок 2.13 – Вікно створення головного класу програми архівації однотипних файлів

Відмітимо, що в полі «Extends» має бути вказано ім′я батьківського класу програми архівації однотипних файлів (в даному випадку це є JFrame, клас головного вікна програми архівації однотипних файлів), а в списку «Optional Attributes» інтерфейси, які він повинен реалізовувати.

Для вирішення поставлених у даній роботі задач необхідні інтерфейси ActionListener та ItemListener, які дозволяють класу бути обробником, відповідно для подій натискання кнопки і переключення радіокнопки.

Процес конструювання користувацького інтерфейсу програми архівації однотипних файлів можна вести 2 способами – за допомогою візуального редактора, або за допомогою метода «жорсткого кодування». Простіше, звісно, за допомогою редактора (рис. 2.14).

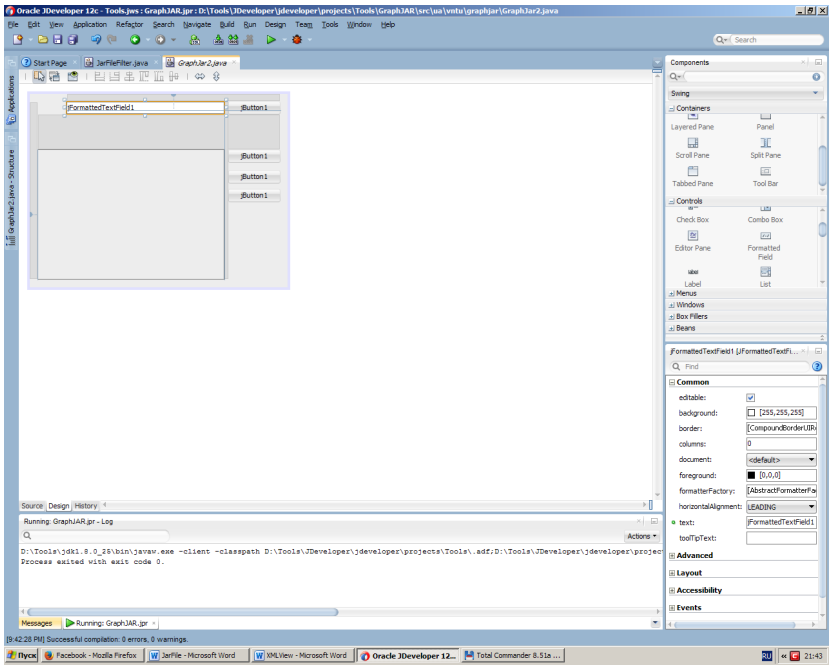


Рисунок 2.14 – Візуальний редактор форм JDeveloper

Остаточний дизайн користувацького інтерфейсу програми архівації однотипних файлів виглядатиме ось таким чином (рисунок 2.15):

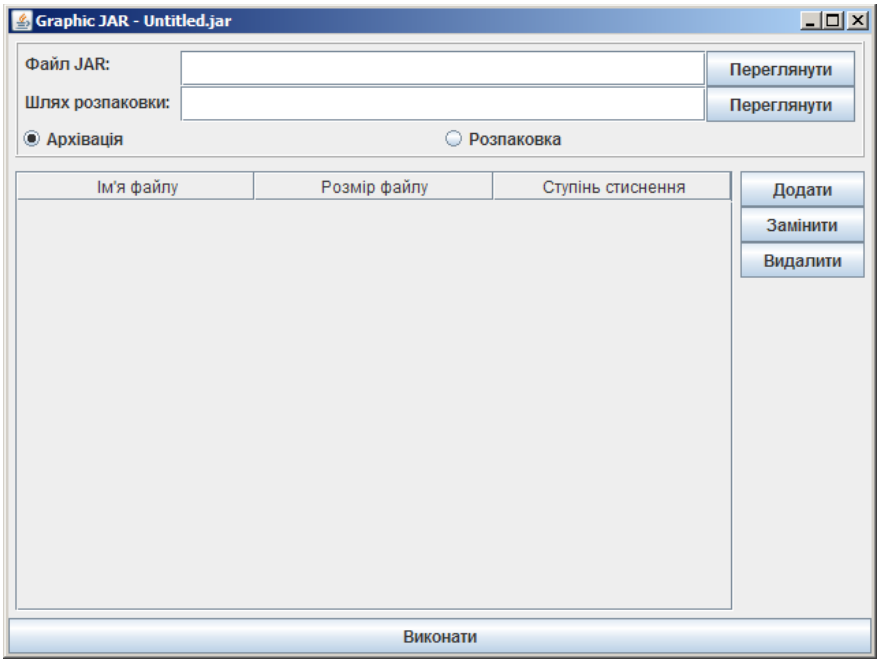


Рисунок 2.15 – Дизайн користувацького інтерфейсу програми архівації однотипних файлів

Під час самої розробки програми архівації однотипних файлів скористаємось одним із спеціальних класів Java з метою створення динамічної панелі, що була би здатна перемикатись між 2 різними панелями – із текстовою зоною та компонентом-деревом. Отож, ми економимо робочий простір програми архівації однотипних файлів, і досягаємо максимуму подачі даних.

Після того, як інтерфейс програми сконструйовано, можна розподілити функції за деякими елементами:

Таблиця 2.2 – Співставлення функцій

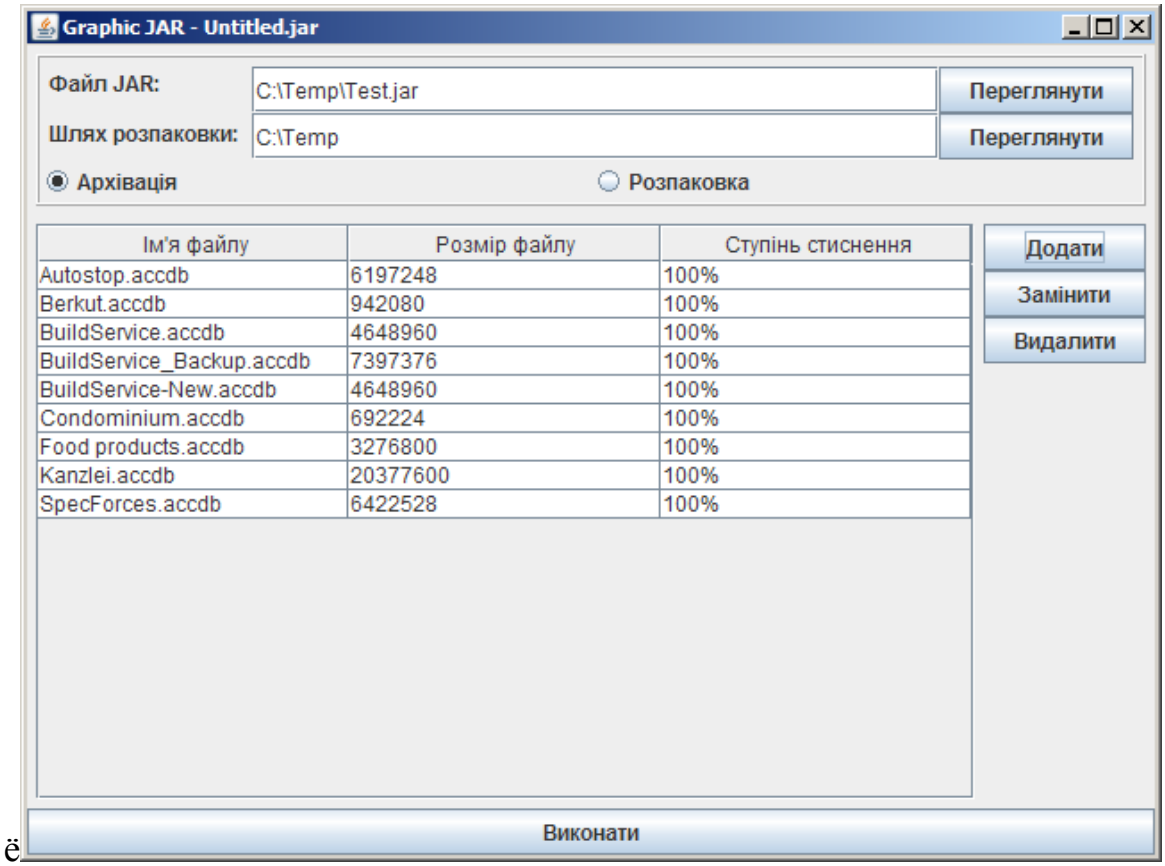
Елемент керування	Функція
Кнопка Переглянути (файл JAR)	Відкриття архіву для процесу стиснення та архівації чи розпаковування.
Кнопка Переглянути (шлях розпаковування)	Вказання каталога, в який поміщуються розпаковані файли
Радіокнопка Архівація	Переключення програми в режим процесу стиснення та архівації
Радіокнопка Розпаковування	Переключення в режим розпаковування
Табличний елемент (склад архіву)	Відображення змісту архіву
Кнопка Додати	Додає новий файл до складу архіву
Кнопка Змінити	Замінює вибраний файл в складі архіву іншим
Кнопка Видалити	Видаляє файл із списку архіву

Реалізація всіх цих функцій і складає простий графічний інтерфейс для програми архівації однотипних файлів. Вихідний текст програми архівації однотипних файлів наведено в додатках.

2.8 Тестування і перевірка правильності роботи програми архівації однотипних файлів

Перевіримо роботу програми архівації однотипних файлів на тестовому прикладі. Запакуємо і розпакуємо десять файлів, а потім порівняємо їх вміст за допомогою спеціальної утиліти порівняння каталогів. Створимо два каталоги, Source та Dest.

Початковий стан програми архівації однотипних файлів виглядатиме так, як на рисунку 2.16.



ё

Рисунок 2.16 – Формування списку файлів до процесу архівації програмою архівації однотипних файлів

Тепер натиснемо кнопку Виконати (потрібний режим уже виставлено (рисунок 2.17).

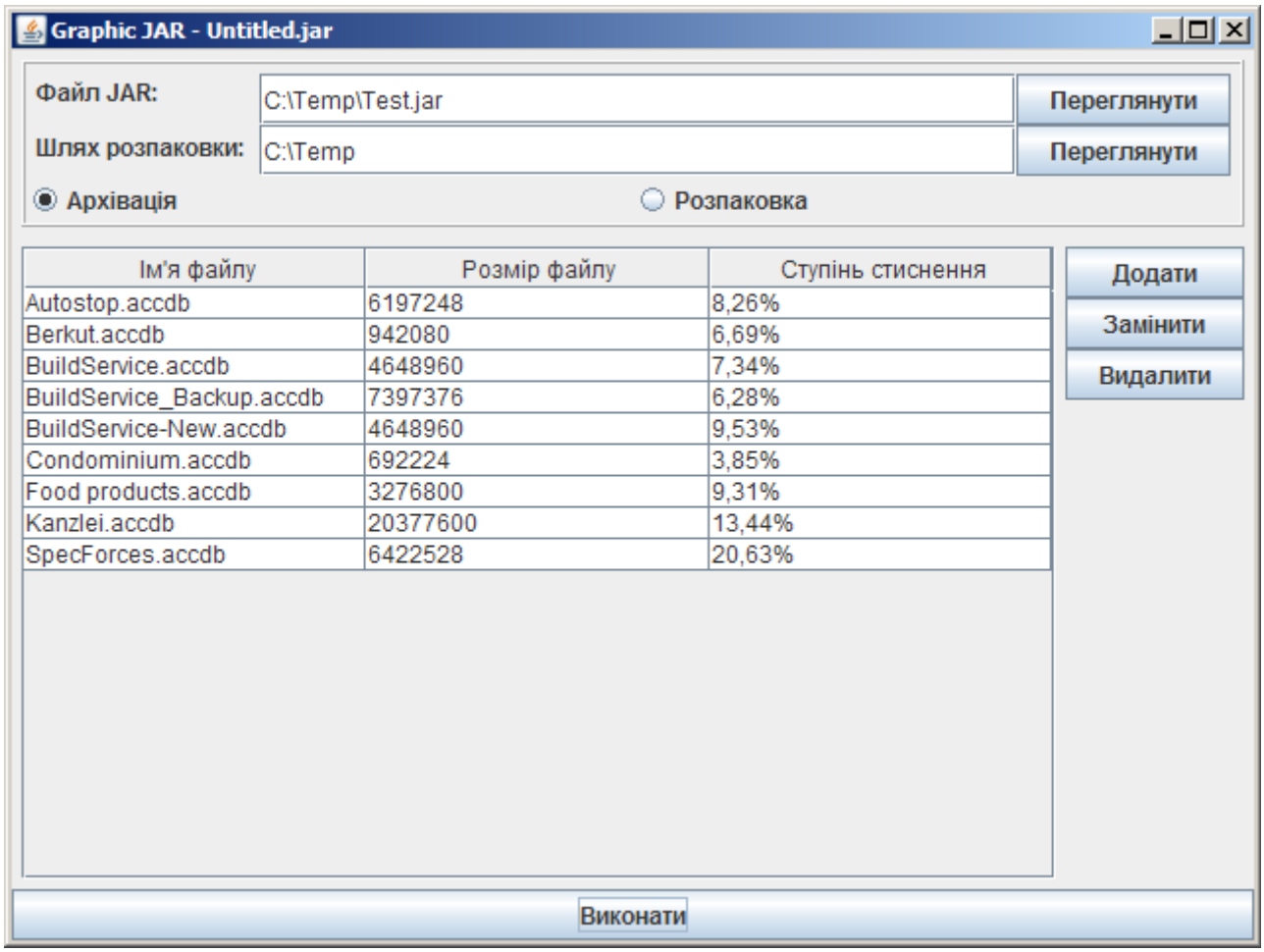


Рисунок 2.17 – Стан програми архівації однотипних файлів після виконання процесу архівації

Графа «Ступінь процесу архівації» змінилась. Вона показує, скільки відсотків від оригінального файлу лишилось після стискання програмою архівації однотипних файлів.

Тепер давайте розпакуємо ці файли в окремий каталог. Для цього заново запустимо програму архівації однотипних файлівё, перемкнемо режим на «Розпаковування» і виберемо створений нами архів. Робочий простір повинен мати такий вигляд (рис. 2.18).

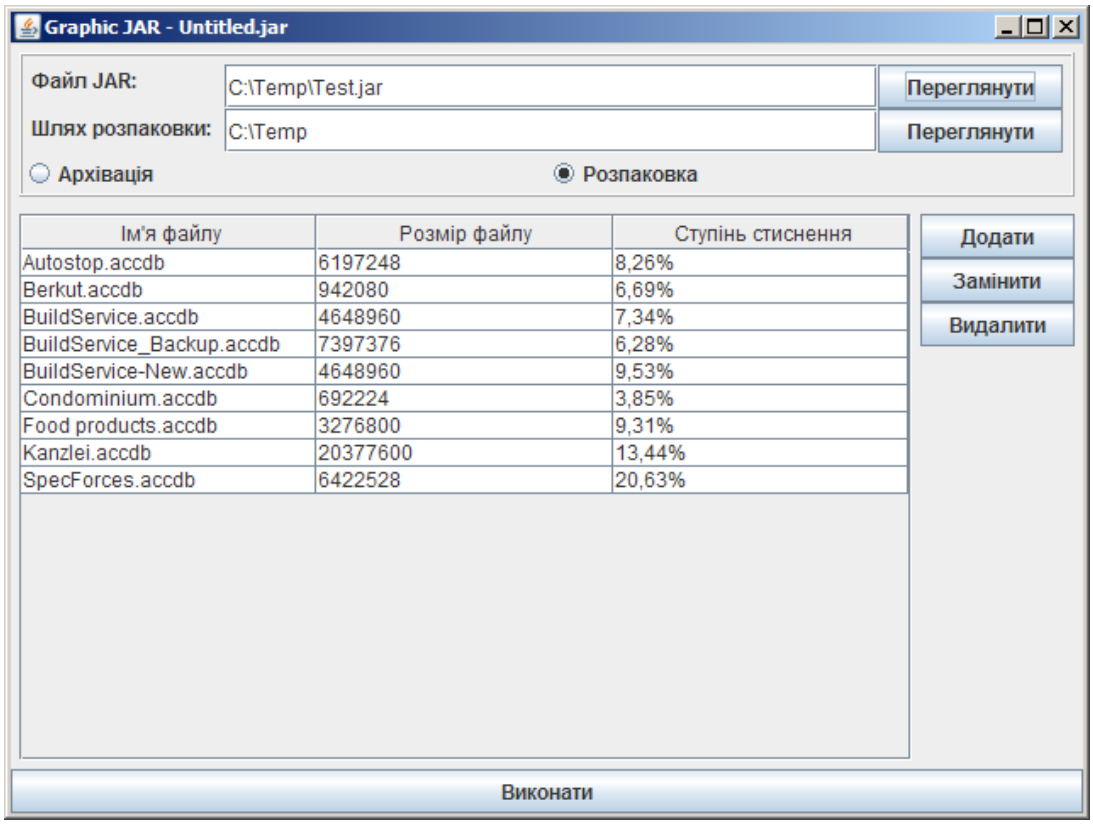


Рисунок 2.18 – Процес відкриття архіва

Для того, щоб розпакувати всі файли, просто не треба вибирати жодного файлу в таблиці списку. Тепер порівняємо каталоги «Source» та «Dest» за допомогою утиліти порівняння каталогів «Beyond Compare», виставивши опції двійкового порівняння (Hex Compare), і показ тільки ідентичних файлів (Same).

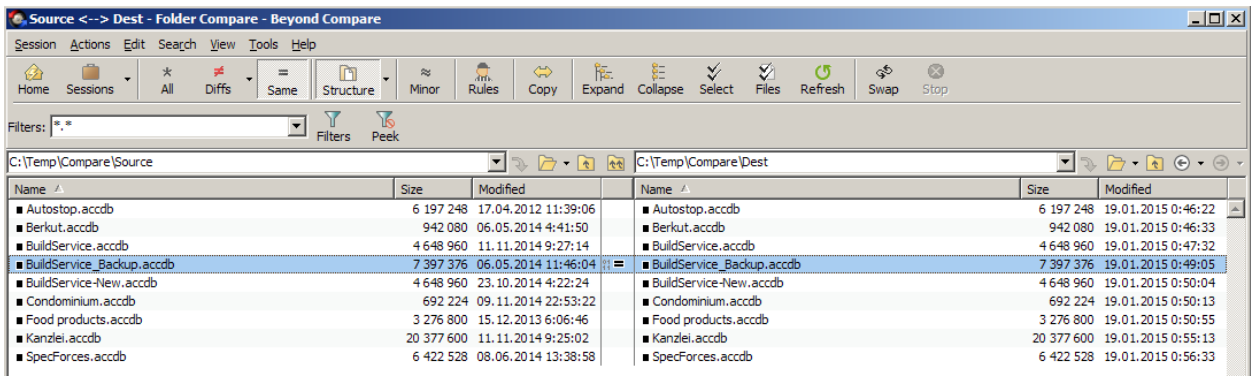


Рисунок 2.19 – Порівняння оригінального і розпакованого каталога

Тепер визначимо ефективність архіватора в залежності від типу файлів. Перевіримо середній показник стискання для різних форматів файлів.

Таблиця 2.3 – Результат роботи процесу стиснення та архівації

Тип файлів	Середнє процесу стиснення та архівації , %
Документ (*.docx)	92,875
Документ (*.doc)	37,7825
Текстовий (*.txt)	55,305
Бази даних (*.accdb)	18,89
Графічні файли (*.jpg)	99,31
Звукові файли (*.mp3)	97,4
HTML (*.html)	14,74

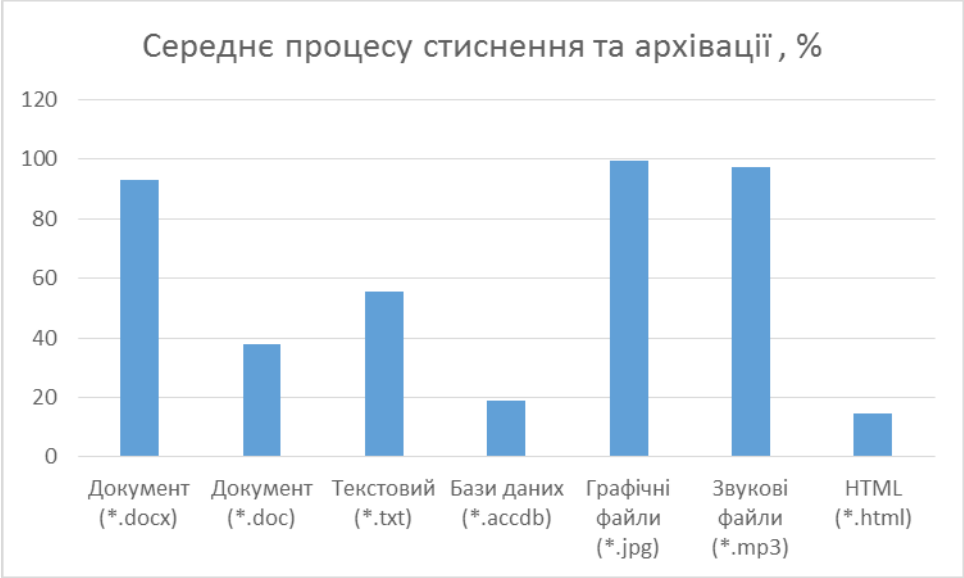


Рисунок 2.20 – Робота процесу стиснення та архівації

З цієї таблиці очевидно, що спеціально розроблений і програмно реалізований алгоритм для методу архівації однотипних файлів найкраще працює для файлів із великою кількістю повторюваних даних (формати БД, чи веб-сторінок). Тому саме для цих категорій файлів його доцільно використовувати, причому бажано файлів повинно бути декілька – тоді кількість повторюваних даних більша, а відтак і процесу стиснення та архівації більше.

2.9 Класи та функції програми архівації однотипних файлів

До складу програми архівації однотипних файлів та супроводу її графічного інтерфейсу входить один базовий клас і один допоміжний.

- GraphJar. Базовий клас програми, успадкований від стандартного класу головного вікна програми JFrame. Базовий клас GraphJar, власне, і реалізує інтерфейси для ActionListener та для ItemListener. Являє собою головний клас програми архівації однотипних файлів, який і виконується при запуску програми.
- JarFileFilter. Успадкований від стандатрного класу FileFilter. Допоміжний клас JarFileFilter дозволяє під час вибору файлів відфільтрувати лише каталоги та JAR-документи.

Діаграма відношень між цими класами та використаними стандартними компонентами показана на рис. 2.21.

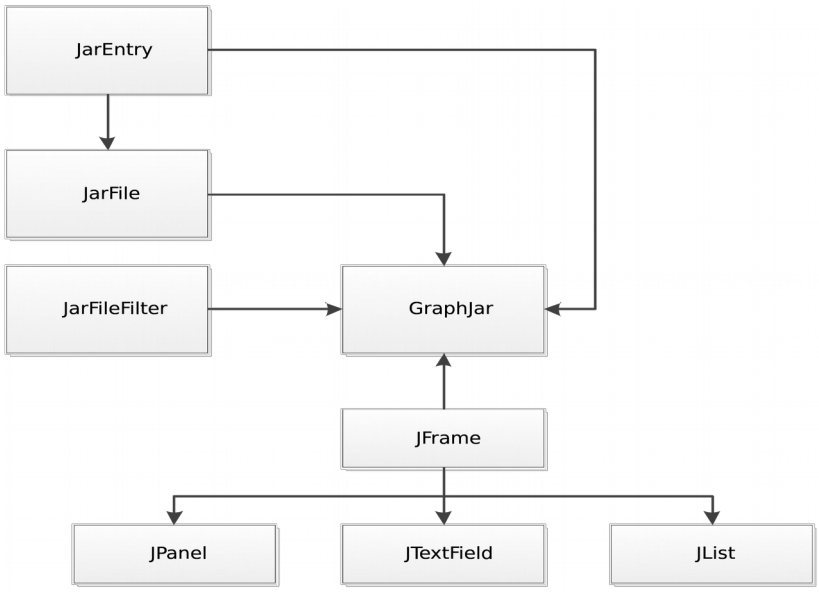


Рисунок 2.21 – Діаграма класів програми архівації однотипних файлів

Кожен із класів програми архівації однотипних файлів має свої властивості та функції. В таблиці 2.4 наведені всі властивості, які має кожен із них, та їх функції:

Таблиця 2.4 – Властивості класів

Назва властивості	Тип	Значення
Клас GraphJar		
currentArchive	String	Поточний архів
jarFile	JTextField	Визначає шлях до архіву. Застосовується як назва створюваного архіву, або файлу, із якого необхідно вести розпакування.
unpackPath	JTextField	Визначає шлях, за яким розміщуються розпаковані файли.
flagPack	JRadioButton	Прапор, що ініціює режим процесу стиснення та архівації.
flagUnPack	JRadioButton	Прапор, що ініціює режим розпакування.
fileList	JTable	Список файлів, що підлягають архівуванню або розпакуванню.

Клас JarFileFilter спеціальних полів не містить, оскільки є допоміжним. Зведемо також до таблиці 2.5 і опишемо функції, із яких складається кожний із вищеперелічених класів.

Таблиця 2.5 – Функції класів, що входять до програми

Назва функції	Список параметрів	Опис
Клас XMLFileFilter		
accept	File f	Повертає булевев значення, визначає чи задовольняє даний файл критерію фільтрування. Автоматично викликається компонентом JFileChooser.
getDescription		Повертає строковий опис фільтру, скажімо, «JAR Documents»
Клас GraphJar		
GraphJar		Конструктор головного вікна програми архівації однотипних файлів, що створює користувацький інтерфейс і забезпечує виконання алгоритму роботи програми.
createButton	String title String command	Створює кнопку із потрібною назвою і співставляє йому конкретну команду.
getBytesFromFile	String canonPath	Перетворює вхідний файл на потік байтів.
clearTable		Видаляє всі файли, які занесено в список архівування.
createJarFile		Створює архівний файл за списком даних файлів.
openJarFile	String what	Відкриває архівний файл і заповнює таблиці вмістом архіву.
extractFromJar	String what String name	Функція, що розпаковує конкретний файл із jar-архіву.
actionPerformed	ActionEvent e	Обробник, що відповідає за дії, прив’язані до подій натиснення

		кнопок, тобто, вибір файлів, запуск процедури архівування та розпаковування тощо.
itemStateChanged	ItemEvent e	Обробник радіокнопок, відповідає за перемикання режимів процесу стиснення та архівації і розпаковування.
main	String[] args	Точка входу в програму архівації однотипних файлів, відображає головне вікно програми.

Таким чином, створена програма повністю реалізовує поставлену задачу побудови графічного інтерфейсу для програми архівації файлів.

2.10 Висновки за розділом

У даному розділі розглянуто питання практичного застосування кросплатформеної технології Java з точки зору питань архівування. Головна увага приділена розробці та опису запропонованого методу архівації однотипних файлів, а також, програмній реалізації цього методу засобами Java.

Зокрема, під час розробки методу архівації однотипних файлів розроблено процеси формування початкового та основного словників, процесу архівації, та розроблено алгоритму роботи кросплатформеного засобу архівації однотипних файлів, побудовано користувацький інтерфейс для програми архівації однотипних файлів та наведено діаграму класів.

Виконано тестування і перевірка правильності роботи програми архівації однотипних файлів.

3 ЕКОНОМІЧНА ЧАСТИНА

3.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності [49].

Результатом магістерської кваліфікаційної роботи «Метод та кросплатформений засіб архівації однотипних файлів» є розробка методу архівації однотипних файлів, вдосконалено процес формування основного словника, вдосконалено процес архівації файлів. Для проведення технологічного аудиту залучено трьох незалежних експертів. У нашому випадку такими експертами є: : Колесник Ірина Сергіївна (к.т.н., доцент каф. обчислювальної техніки ВНТУ), Снігур Анатолій Васильович (к.т.н., доцент каф. обчислювальної техніки ВНТУ) та Захарченко Сергій Михайлович (к.т.н., доцент каф. обчислювальної техніки ВНТУ).

Оцінювання комерційного потенціалу буде здійснене за критеріями, що наведені в таблиці 3.1.

Таблиця 3.1 - Критерії оцінювання комерційного потенціалу розробки бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження таблиці 3.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)

Кри-тер.	0	1	2	3	4
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження таблиці 3.1

11	Термін	Термін	Термін	Термін	Термін реалізації

	реалізації ідеї більший за 10 років	реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання комерційного потенціалу зведено в таблицю 3.2.

Таблиця 3.2 - Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	1 – Колеснік	2 – Снігур	3 – Захарченко
	Бали, виставлені експертами:		
1	4	3	4
Ринкові переваги (недоліки):			
2	3	2	2
3	4	4	4
4	4	3	4
5	3	4	3
Ринкові перспективи			
6	2	2	2
7	3	4	3
Практична здійсненність			
8	4	3	4
9	3	2	3
10	4	4	4
11	3	4	3
12	3	3	3
Сума балів	$СБ_1$=40	$СБ_2$=38	$СБ_3$=39
Середньоарифметична сума балів $\overline{СБ}$	39		

За даними таблиці 3.2 можна зробити висновок, щодо рівня комерційного потенціалу розробки. Зважимо на результат й порівняємо його з рівнями комерційного потенціалу розробки, що представлено в таблиці 3.3.

Таблиця 3.3 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів $\overline{СБ}$, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 48	Високий

Рівень комерційного потенціалу розробки, становить 39 балів, що відповідає рівню «вище середнього».

Задача компактного зберігання, перетворення та передавання інформаційних даних завжди була актуальною в галузі інформаційних технологій.

Інформаційні ресурси нині є продуктом інтелектуальної діяльності дійсно найбільш кваліфікованої й творчо активної частини молоді та працездатного населення світу. В останній чверті XX століття набуті інформаційні ресурси досягли (і продовжують досягати) настільки рекордних обсягів, що цілком повсякденним стали поняття «інформаційного вибуху», «інформаційної революції». В якості доказу є об’єктивне збільшення інформаційного потоку з початку цього сторіччя більш ніж в 30 разів.

З метою забезпечити надійне збереження інформації створюють резервні копії даних. Задача збереження резервних копій у компактному вигляді є основою для процесів архівації та стиснення даних. В загальному випадку, основний зміст архівації полягає у створенні таких резервних копій, які потребували би значно меншого обсягу на інформаційних ресурсах, ніж та сама інформація у вихідному стані. Таким чином, в контексті під архівацією слід розуміти процес перекодування деякої сукупності файлів з метою зменшення загального об’єму пам’яті, який вони займають. Часто архівацією ще називають процес стиснення даних.

Нині відомо досить багато різних підходів до процесу архівації. Усі підходи мають в своїй основі різні підходи та різні методи, проте подібні вони в одному – це те, що вони сповідують принцип заміни рівномірного

двійкового коду на нерівномірний. З метою архівації файлів та папок використовують спеціальні програмні засоби, які називають архіваторами. Стиснуті файли поміщають у файли, який називають архівам.

Основними можливостями сучасних архіваторів є такі:

- занесення груп файлів та (або) підкаталогів в архів;
- можливість поновлення архіву;
- перегляд файлів з меж архіву;
- вилучення окремих файлів з архіву;
- захист файлів від несанкціонованого доступу (НСД);
- перевірка архіву на цілісність;
- створення багатотомних архівів;
- можливість створення архівів, що автоматично відкриваються

Можливості сучасних програм-архіваторів дозволяють зекономити від 20 до 90 відсотків дискового простору. Файлом, що знаходиться в архіві, можна скористатися після того, як він буде відновлений у початковому вигляді, тобто розархівований (розпакований). Розархівування виконують або ті ж самі програми-архіватори в зворотному напрямку, або окремі програми, які називають розрахіваторами, серед яких найбільш відомими є: ZIP, JAR, RAR.

Під час вибору конкретного засобу для архівування (розархівування) користувачі керуються багатьма критеріями, як то швидкістю роботи, коефіцієнти стискування даних, інтерфейс, сумісність тощо. Важливим є те, що для одного типу файлів кращим може бути один архіватор, а для іншого – інший.На сьогодні вже розроблено ядро програмного продукту, яке містить основний функціонал. В подальшому планується виконати оптимізацію роботи програми. Для здійснення розробки не потрібна наявність фінансових ресурсів, а лише фахівців відповідної кваліфікації.

Для подальшого вдосконалення розробки може знадобитись залучення фахівців з відповідних технологій. Розроблена програма є готовим програмним продуктом, але при потребі вона може бути розширена та оптимізована для використання у конкретній сфері.

Для просування продукту на ринок необхідно провести рекламну кампанію та розробити власний сайт для можливості поширення серед потенційних клієнтів готового продукту.

Для усіх користувачів розробки на сайті продукту буде доступний зворотній зв’язок з розробником, куди можна буде звертатись з усіх питань, які стосуються нової розробки.

3.2 Прогнозування витрат на виконання науково-дослідної роботи та впровадження її результатів.

Прогнозування витрат на виконання науково-дослідної роботи складається з таких етапів: розрахунок витрат, які безпосередньо стосуються виконавців даного розділу роботи; розрахунок загальних витрат на виконання даної роботи; прогнозування загальних витрат на виконання та впровадження результатів даної роботи.

Виконаємо розрахунок витрат, які безпосередньо стосуються виконавця даного розділу роботи, приймаючи до уваги те, що розробкою займався один розробник.

Основна заробітна плата для розробника розраховується за формулою:

$$З_о = \frac{М}{Т_р} \cdot t, [грн], \quad (3.1)$$

де М - місячний посадовий оклад розробника, М = 12000 грн.;
$Т_р$ – кількість робочих днів у місяці, $Т_р$ = 21 день;
t - число днів роботи розробника, t = 66 дні.

$$З_о = \frac{12000{,}00}{21} \cdot 66 = 37714{,}28\ (грн).$$

Додаткова заробітна плата розробника, що брав участь у виконанні даного етапу роботи, розраховується як 10 % від суми основної заробітної плати розробника за формулою:

$$З_д = 0,10 \cdot З_о \text{ [грн]}. \quad (3.2)$$

$$З_д = 0,10 \cdot 37714,28 = 3771,42 \text{ (грн)}.$$

Нарахування (ЄСВ) на заробітну плату ($Н_{зп}$) розробника, що брав участь у виконанні даного етапу роботи становлять 22%. Нарахування на заробітну плату розробника розрахуємо за формулою:

$$H_{зп} = (З_о + З_р) \cdot \frac{\beta}{100} \text{ [грн]}, \quad (3.3)$$

де $З_о$ – основна заробітна плата розробників, грн;

$З_д$ – додаткова заробітна плата всіх розробників та робітників, грн;

β – ставка єдиного соціального внеску на загальнообов'язкове державне соціальне страхування, %.

$$Н_{ЗП} = (37714,28+3771,42) \cdot 0,22 = 9126,85 \text{ (грн)}.$$

Амортизація обладнання та приміщення, яке використовувалось для проведення розробки, розраховується за формулою:

$$A = \frac{Ц \cdot H_a}{100} \cdot \frac{T}{12} \text{ [грн]}, \quad (3.4)$$

де Ц – балансова вартість обладнання, грн.;

$Н_а$ – річна норма амортизаційних відрахувань;

Т – термін використання під час розробки, місяців.

Норма амортизації розраховується за формулою:

$$H_a = \frac{B_п - B_л}{B_п \cdot T_{кв}} \cdot 100 \text{ [грн]}, \quad (3.5)$$

де $В_п$ і $В_л$ – відповідно первісна та ліквідаційна вартість основних

фондів;

$Т_{кв}$ – строк корисного використання, 5 роки.

$$H_a = \frac{20000 - 2000}{20000 \cdot 5} \cdot 100 = 18\% \text{ (грн)}.$$

Розрахуємо амортизаційні витрати на ноутбук, балансова вартість якого становить 20000 грн, а термін використання – 3 місяці:

$$A = \frac{20000 \cdot 18}{100} \cdot \frac{3}{12} = 900 \text{ (грн)}.$$

Зроблені розрахунки зведено до таблиці 5.5.

Таблиця 3.5 – Амортизаційні відрахування

Найменування	Балансова вартість, грн	Термін використання, р	Фактична трив. використання, міс.	Величина амортизаційних відрахувань, грн
Приміщення	380000,00	20	3	4275,00
Ноутбук	20000,00	5	3	900,00
Всього:				5175,00

Всі види витрачених матеріалів їх вартість і кількість відображені в таблиці 3.6.

Таблиця 3.6 – Витрати на матеріали, що були використані для розробки.

Найменування матеріалу	Одиниці виміру	Ціна за одиницю, грн	Витрачено, шт	Вартість витрачених матеріалів, грн
Папір	уп.	90	1	100,00
Ручка	шт.	10	2	20,00
Олівець	шт.	13	1	15,00
Загальна сума витрат за статтею				135,00

Витрати на силову електроенергію ($В_е$) для виконання даного етапу роботи, розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п, [грн], \quad (3.6)$$

де $В$ – вартість 1 кВт-год. електроенергії, становить 2,4 грн./кВт.

$П$ – установлена потужність комп'ютера, кВт (у нашому випадку становить 0,3 кВт);

$Ф$ – фактична кількість годин роботи обладнання.

$К_п$ – коефіцієнт використання потужності, становить 0,3.

$$В_е = 2,4 \cdot 0,3 \cdot 528 \cdot 0,3 = 114,00 \text{ (грн)}.$$

Інші витрати ($В_{ін}$) охоплюють: витрати на управління організацією, оплату службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, Інтернет послуги. Інші витрати $І_в$ можна прийняти як 50% від суми основної заробітної плати розробника:

$$В_{ін} = 50\% \cdot (З_р) \text{ [грн]}. \quad (3.7)$$

$$В_{ін} = 0,5 \cdot 37714,28 = 18857,14 \text{ (грн)}.$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини (розділу, етапу) роботи – В:

$$В = 37714,28+3771,42 +9126,85 +5175+135,00+114,00+18857,14=78451,84 \text{ (грн)}.$$

Наступний етап роботи полягає у розрахунку загальних витрат на виконання даної роботи $В_{заг}$ та визначається за формулою:

$$В_{заг} = \frac{В}{\alpha} \text{ [грн]}, \quad (3.8)$$

де α – частка витрат, які безпосередньо здійснює виконавець даного

етапу роботи, у відн. одиницях (α= 1.)

$$В_{заг.} = \frac{78451,84}{1} = 78451,84 \quad (грн).$$

Далі визначаємо загальні витрати на виконання та впровадження результатів виконаної наукової роботи (ЗВ) за формулою:

$$ЗВ = \frac{В_{заг}}{\beta} [\text{ грн}], \qquad (3.9)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

$$ЗВ = \frac{78451,84}{0,9} = 87169,00 \ (грн).$$

Витрати на виконання наукової роботи та впровадження її результатів становитиме 87169,00 грн.

3.3 Прогнозування комерційних ефектів від реалізації результатів розробки

У даному підрозділі проведемо кількісне прогнозування, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. В умовах ринку узагальнюючим позитивним результатом, що його отримує підприємство від впровадження результатів тієї чи іншої розробки, є збільшення чистого прибутку підприємства. Виконання даної наукової роботи та впровадження її результатів складає приблизно 1 рік. Позитивні результати від впровадження розробки очікуються вже в перші місяці після впровадження. Проведемо детальніше прогнозування позитивних результатів та кількісне їх оцінювання по роках. Обчислимо збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_i = \sum_1^n (\Delta П_я \cdot N + П_я \cdot \Delta N)_i \ [грн], \qquad (3.10)$$

де $\Delta П_я$ – покращення основного якісного показника від впровадження результатів розробки у даному році;

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ – основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

Припустимо, що внаслідок впровадження результатів наукової розробки чистий прибуток підприємства збільшиться на 50,00 грн, а кількість одиниць реалізованої послуги збільшиться: протягом першого року – на 3000 од., протягом другого року – ще на 5000 од., протягом третього року – ще на 6000 од. Орієнтовно: реалізація продукції до впровадження результатів наукової розробки складала 1 шт., а прибуток, що його отримувало підприємство на одиницю продукції до впровадження результатів наукової розробки – 30,00 грн. Потрібно спрогнозувати збільшення чистого прибутку підприємства від впровадження результатів наукової розробки у кожному році відносно базового.

Збільшення чистого прибутку підприємства $\Delta П_1$ протягом першого року складе:

$$\Delta П_1=30\cdot 1+(30+50)\cdot 3000=400030{,}00 \text{ (грн)}.$$

Обчислимо збільшення чистого прибутку підприємства $\Delta П_2$ протягом другого року:

$$\Delta П_2=30\cdot 1+(30+50)\cdot (3000+5000)=640030{,}00 \text{ (грн)}.$$

Збільшення чистого прибутку підприємства $\Delta П_3$ протягом третього року становитиме:

$$\Delta П_3=30\cdot 1+(30+50)\cdot (3000+5000+6000)=1120030{,}00 \text{ (грн)}.$$

Отже, розрахунки показують, що комерційний ефект від впровадження розробки виражається у значному збільшенні чистого прибутку підприємства.

3.4 Розрахунок ефективності вкладених інвестицій та період їх окупності

Розрахований комерційний ефект від можливого впровадження розробок ще не означає, що ця розробка реально буде впроваджена. Якщо збільшення прогнозованого прибутку від впровадження результатів наукової розробки є вигідним для підприємства (організації), то це ще не означає, що інвестор погодиться фінансувати дану розробку. Інвестор погодиться вкладати кошти у реалізацію даної наукової розробки тільки за певних умов. Основними показниками, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності. Розрахунок ефективності вкладених інвестицій передбачає проведення таких робіт:

1. Розраховуємо теперішню вартість інвестицій PV, що вкладаються в наукову розробку. Будемо вважати, загальні витрати на виконання та впровадження результатів НДДКР (або теперішня вартість інвестицій PV) дорівнює 87169,00 грн.

2. Очікуване збільшення прибутку $\Delta П_i$, що його отримає підприємство від впровадження результатів наукової розробки, для кожного із років: у першому році підприємство отримає збільшення чистого прибутку відносно базового на 400030,00 грн, у другому році – на 640030,00 грн, у третьому році – на 1120030,00 грн.

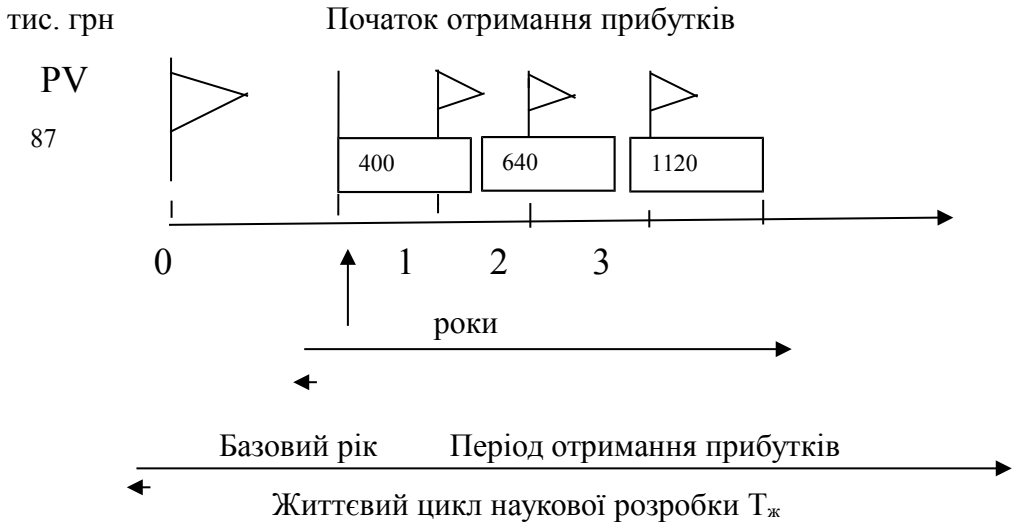


Рисунок 3.1 – Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

3. На третьому етапі, будуємо вісь часу, на яку наносимо всі платежі (інвестиції та прибутки), що мають місце під час виконання НДДКР та впровадження її результатів.

Наступним кроком є розрахунок абсолютної ефективності вкладених інвестицій $E_{абс}$:

$$Е_{абс} = (ПП - PV), \text{ [грн]}, \quad (3.11)$$

де ПП – приведена вартість всіх чистих прибутків, що їх отримає підприємство (організація) від реалізації результатів наукової розробки, грн.;

PV – теперішня вартість інвестицій PV = ЗВ, грн.

У свою чергу, приведена вартість всіх чистих прибутків ПП розраховується за формулою:

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \text{ [грн]}, \quad (3.12)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки 0.

$$ПП = \frac{400030{,}00}{(1+0{,}1)^1} + \frac{640030{,}00}{(1+0{,}1)^2} + \frac{1120030{,}00}{(1+0{,}1)^3} = 1734741{,}86(\text{грн}).$$

Після цього розраховуємо абсолютної ефективності вкладених інвестицій $E_{абс}$:

$$E_{абс} = 1734741{,}86 - 87169{,}00 = 1647572{,}86(\text{грн.})$$

Оскільки $E_{абс} > 0$, то результат від проведення наукових досліджень та їх впровадження принесе прибуток, а вкладання коштів у даний проект є доцільним.

Розраховуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$ за формулою:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1 \quad (3.13)$$

де $Е_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн;
$T_ж$ – життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[3]{1 + \frac{1647572,86}{87169,00}} - 1 = \sqrt[3]{19,9} - 1 = (2,70 - 1) = 1,70 \text{ або } 170\%$$

Далі, розраховану величину $E_в$ порівнюємо з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau = d + f, \tag{3.14}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; d = 0,2;

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1), але може бути і значно більше, у нашому випадку f = 0,5.

$$\tau = 0,2 + 0,05 = 0,25$$

Оскільки $E_в$ = 170 % > $\tau_{мін}$ = 25%, то інвестор буде зацікавлений вкладати гроші в дану наукову розробку, адже він отримає значно більші прибутки, ніж якщо просто покладе свої гроші на депозит у комерційному банку.

Наступним кроком є визначення терміну окупності вкладених у реалізацію наукового проекту інвестицій. Термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{ок}$ розраховується за формулою:

$$T_{ок} = \frac{1}{E_в} [\text{року}] \tag{3.15}$$

Отже, термін окупності складе:

$$T_{ок} = \frac{1}{1,70} = 0,59(року).$$

Термін окупності інвестицій $T_{ок}$ 0,59 < 3…5 років і свідчить, що фінансування даної наукової розробки є доцільним.

Отже, розрахунок ефективності вкладених інвестицій та періоду їх окупності показав, що фінансування розробки є доцільним, оскільки інвестиції буде повернуто в термін до одного року.

3.5 Висновок

В даному розділі було виконано оцінювання комерційного потенціалу нової розробки. Проведено технологічний аудит з залученням трьох незалежних експертів. Визначено, що рівень комерційного потенціалу розробки вище середнього. Аналіз комерційного потенціалу розробки показав, що програмний продукт за своїми характеристиками випереджає аналогічні програмні продукти, що підтверджує її перспективність.

Згідно із розрахунками всіх статей витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторсько-технологічної роботи загальні витрати на розробку складають 87169,00 грн.

Розрахована абсолютна ефективність вкладених інвестицій в сумі 1647572,86 грн свідчить про отримання прибутку інвестором від комерціалізації програмного продукту.

Щорічна ефективність вкладених в наукову розробку інвестицій складає 170 %, що вище за мінімальну бар'єрну ставку дисконтування, яка складає 25%. Це означає потенційну зацікавленість інвесторів у фінансуванні розробки.

Термін окупності вкладених у реалізацію проекту інвестицій становить 0,59 року, що також свідчить про доцільність фінансування нової розробки.

ВИСНОВКИ

Підсумком виконання даної магістерської кваліфікаційної роботи стала розробка методу та кросплатформеного засобу архівації однотипних файлів. Високий рівень виконання поставленої прикладної задачі вирішено засобами мови програмування Java.

Розроблений метод призначений для процесів оптимізованої архівації великих кількостей маленьких однотипних файлів. В основі запропонованого роботи методу архівації однотипних файлів є ідея заміни повторного входження цілого блока даних посиланням на попередню позицію його входження.

Зокрема, у роботі отримані такі наукові результати:

вперше запропоновано метод архівації однотипних файлів, який дозволяє збільшити середнє значення процесу архівації однотипних файлів на 10, 85%;

вдосконалено процес формування основного словника за рахунок застосування методу «ковзного вікна» до його формування;

вдосконалено процес архівації за рахунок застосування вдосконаленого процесу формування основного словника.

Практичне значення одержаних результатів полягає у такому: розроблено новий метод архівації однотипних файлів, вдосконалено процес формування основного словника за рахунок застосування до його формування методу «ковзного вікна»; розроблено алгоритм роботи формування основного словника однотипних файлів, який дозволяє ефективніше стискати велику кількість однотипних файлів; розроблено програмний засіб для архівації однотипних файлів.

В рамках даної роботи виявлено, що розроблений метод архівації однотипних файлів гарно працює в умовах наявності не менше 10 файлів на вході методу, а при наявності більшої кількості (100 і більше) однотипних файлів значення Arh_m росте. Запропонований метод дає приріст середнього значення процесу архівації на 10,85%, а зі збільшенням кількості однотипних файлів для процесу архівації цей показник може все більше зростати.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Семѐнов Ю. Телекоммуникационные технологи / Семенов Ю. – М.: Бином. Лаборатория знаний, 2007. – 640с.
2. Сергеенко В. Сжатие данных, речи, звука и изображений в телекоммуникационных системах / Сергеенко В., Баринов В. – М.: КудицОбраз, 2009. – 360с.
3. Дейт К. Введение в системы баз данных. – 7-е изд. / Дейт К. – М.: Вильямс,
4. 2001. – 1072 с.
5. Кнут Д. Искусство программирования. Том 1. Основные алгоритмы / Кнут Д. – М.: Вильямс, 2010. – 720с.
6. Кокс Д. Идеалы, многообразия и алгоритмы. Введение в вычислительные
7. аспекты алгебраической геометрии и коммутативной алгебры / Кокс Д., Литтл Д. – М.: Мир, 2000. – 688с.
8. Коллинз Д. Структуры данных и стандартная библиотека шаблонов / Коллинз Д. – М.: Беном-Пресс, 2003. – 624с.
9. Кормен Т. Алгоритмы. Построение и анализ / Кормен Т., Лейзерсон Ч., Ривест Р. – М.: Вильямс, 2011. – 1296с.
10. Красильников Н. Цифровая обработка 2D- и 3D-изображений / Красильников Н. – СПб.: БХВ-Петербург, 2011. – 608с.
11. Левитин А. Алгоритмы: введение в разработку и анализ / Левитин А. — М.: Вильямс, 2006. — С. 392—398.
12. Макконнелл С. Совершенный код / Макконнелл С. – СПб.: Питер, 2007. – 896с.
13. Марков А. Базы данных. Введение в теорию и методологию / Марков А. - М.: Финансы и статистика, 2006. – 512с.
14. Л. А. Савицька, Т. І. Коробейнікова, П.В. Чирва. Метод та кросплатформений засіб архівації однотипних файлів. Інформаційні

технології та комп'ютерна інженерія – В. : ВНТУ, 2019 – №3 (46). Подана до друку.

15. Молдовян А. Криптография / Молдовян А., Советов Б. – М.: Лань, 2000. – 224с.
16. Морелос-Сарагоса Р. Искусство помехоустойчивого кодирования. Методы, алгоритмы, применение / Морелос-Сарагоса Р. – М.: Техносфера, 2006. – 320с.
17. Новиков Ф. Дискретная математика для программистов / Новиков Ф. – СПб.: Питер, 2001. – 304с.
18. Онопко Є.В. Системи обробки інформації. Випуск 7(97) Блочностатистичний метод стиснення інформації / Онопко Є.В., Тарасов О. В.– Х.: ХУПС, 2011. – 172 с.
19. Онопко Є.В. Системи обробки інформації. Випуск 4(102). Дослідження ефективності блочно-статистичного методу стиснення інформації / Онопко Є.В., Тарасов О. В.– Х.: ХУПС, 2012. – 232 с.
20. Ричардсон Я. Видеокодирование. H.264 и MPEG-4 – стандарты нового поколения / Ричардсон Я. – М.: Техносфера, 2005. – 368с.
21. Селомон Д. Сжатие данных, изображений и звука. / Селомон Д. – М.: Техносфера, 2004. – 368 с.
22. Уоррен Г. Алгоритмические трюки для программистов / Уоррен Г. – М.: Вильямс, 2007. – 288с.
23. Швецов В. Бази даных / Швецов В. – М.: Apress – 2008. – 337 с.
24. Altova Authentic Browser Edition Reference Manual – Altova Press, 2005.
25. Altova Authentic Desktop Edition Reference Manual – Altova Press, 2005.
26. Altova MapForce User's Guide and Reference Manual – Altova Press, 2005.
27. Altova Stylevision User's Guide and Reference Manual – Altova Press, 2005.
28. Altova XML Spy User's Guide and Reference Manual – Altova Press, 2005.
29. Applied XML Solutions – McMillan Computer Pub, 2002
30. B. Eckel Thinking in Java – NY, Prentice Hall, 2000
31. B. McLaughlin, J. Edelson Java and XML – O'Reilly, 2003
32. Code of Virginia, Title 18.2, Chap. 5 Crimes Against Property, Article 7.1. Computer Crimes.

33. D. Benyon, D. Stone, M. Woodroffe Experience with Developing Multimedia Courseware for the World Wide Web – //Human-Computer Studies, 47/1997.
34. E. R. Harold, W. Scott Means XML in a Nutshell – O'Reilly, 2005
35. E. van der Vlist. XML Schema – O'Reilly, 2002
36. G. Weber, M. Specht User Modeling and Adaptive Navigation Support in WWW-Based Tutoring Systems – University of Trier, Dept. of Psychology, NY, 1997.
37. J. Roschelle, C. DiGiano, M. Koutlis, A. Repenning, J. Philips, N. Jackiw, D. Suthers Developing Educational Software Components – // Educational Computing Research, 2001.
38. J. Shirazi Java Performance Tuning – Sebastopol, O'Reilly, 2000
39. M. Fernandez, A. Malhotra, J. Marsh, M. Nagy, N. Walsh XQuery 1.0 and XPath 2.0 Data Model – W3C Working Draft, 2005.
40. M. Kay XSL Transformations 2.0 – W3C Working Draft, 2005.
41. P. C. Dibble Real-Time Java Platform Programming – NY, Prentice Hall, 2002
42. S. Boag, D. Chamberlain, M. Fernandez, D. Florescu, J. Robie, J. Simeon XML Query Language – W3C Working Draft, 2005.
43. S. Mangano XSLT Cookbook – O'Reilly, 2000
44. S. Stelting, O. Maassen Applied Java Patterns – NY, Prentice Hall, 2001
45. T. Bray XML Specification 1.0 – W3C Recommendation, 1998.
46. United States Code, TITLE 17 - January 1, 1998. CHAPTER 1 - SUBJECT MATTER AND SCOPE OF COPYRIGHT.
47. URAN (Ukrainian Research and Academic Network), http: uran.net
48. Хосс Джексон, Тод Маркес "JAVA" справочник професионала. Москва: СП ЭКОМ, 2003.
49. Методичні вказівки до виконання студентами-магістрантами економічної частини магістерських кваліфікаційних робіт / Уклад. В. О. Козловський – Вінниця: ВНТУ, 2012. – 22 с.

Додаток А

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ

_____

_
«____» _____ 20___
року

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської дипломної роботи

«МЕТОД ТА КРОСПЛАТФОРМЕНИЙ ЗАСІБ АРХІВАЦІЇ ОДНОТИПНИХ ФАЙЛІВ»

08-023.МДР.016.00.000.ТЗ

Виконав: студент 2 курсу, групи 1КІ-18м

зі спеціальності:
123 «Комп'ютерна інженерія»
(шифр і назва напряму підготовки)

Чирва П.В.
(прізвище та ініціали)

Керівник: Савицька Л. А.
(прізвище та ініціали)

м. Вінниця – 2020 р.

1. Підстава для виконання магістерської дипломної роботи (ДР)

1.1 Задача компактного зберігання, перетворення та передавання інформаційних даних завжди була актуальною в галузі інформаційних техндяку

ологій. Під час вибору конкретного засобу для архівування (розархівування) користувачі керуються багатьма критеріями, як то швидкістю роботи, коефіцієнти стискування даних, інтерфейс, сумісність тощо. Важливим є те, що для одного типу файлів кращим може бути один архіватор, а для іншого – інший. Розроблений метод призначений для процесів оптимізованої архівації великих кількостей маленьких однотипних файлів. В основі запропонованого роботи методу архівації однотипних файлів є ідея заміни повторного входження цілого блока даних посиланням на попередню позицію його входження.

1.2 Наказ про затвердження теми дипломної роботи.

2. Мета і призначення МДР

Метою дослідження магістерської кваліфікаційної роботи є збільшення середнього значення процесу архівації для великої кількості однотипних файлів.

З метою забезпечити надійне збереження інформації створюють резервні копії даних. Задача збереження резервних копій у компактному вигляді є основою для процесів архівації та стиснення даних. В загальному випадку, основний зміст архівації полягає у створенні таких резервних копій, які потребували би значно меншого обсягу на інформаційних ресурсах, ніж та сама інформація у вихідному стані. Таким чином, в контексті під архівацією слід розуміти процес перекодування деякої сукупності файлів з метою зменшення загального об’єму пам’яті, який вони займають. Часто архівацією ще називають процес стиснення даних.

Нині відомо досить багато різних підходів до процесу архівації. Усі підходи мають в своїй основі різні підходи та різні методи, проте подібні вони в одному – це те, що вони сповідують принцип заміни рівномірного двійкового коду на нерівномірний. З метою архівації файлів та папок

використовують спеціальні програмні засоби, які називають архіваторами. Стиснуті файли поміщають у файли, який називають архівами.

Перші прототипи архіваторів з’явилися у 80-х роках минулого сторіччя. Основними можливостями сучасних архіваторів є такі [9]:

- занесення груп файлів та (або) підкаталогів в архів;
- можливість поновлення архіву;
- перегляд файлів з меж архіву;
- вилучення окремих файлів з архіву;
- захист файлів від несанкціонованого доступу (НСД);
- перевірка архіву на цілісність;
- створення багатотомних архівів;
- можливість створення архівів, що автоматично відкриваються.

3. Вихідні дані для виконання МДР

Список технічної літератури, аналіз, вивчення та дослідження сучасних методів, процесів та засобів архівування, технічне завдання на магістерську дипломну роботу.

Для досягнення поставленої мети у МДР необхідно виконати такі завдання:

провести аналіз сучасних методів та засобів архівації даних в галузі інформаційних технологій, виконати їх порівняльну характеристику та сформувати вимоги та обрати й обґрунтувати вибір методу, що задовольняв би меті даної магістерської кваліфікаційної роботи;

розглянути існуючі способи архівації за кросплатформеною технологією Java;

запропонувати метод архівації однотипних файлів згідно мети магістерської кваліфікаційної роботи, розробити ключові процеси роботи методу архівації однотипних файлів та виконати програмну реалізацію запропонованого методу архівації однотипних файлів;

провести тестування програмного продукту та виконати аналіз отриманих результатів.

4. Матеріали, що подаються до захисту МДР

Пояснювальна записка ДР, графічні і ілюстративні матеріали, протокол попереднього захисту МДР на кафедрі, відгук наукового керівника, відзив рецензента, протоколи складання державних екзаменів, анотації до МДР українською та іноземною мовами.

5. Техніко-економічні показники

5.1 Витрати на програмні засоби, що використовуються в ході даної розробки, повинні бути мінімальними.

5.2 Лімітна ціна на програмне забезпечення не повинна перевищувати ціну аналога.

6. Порядок контролю виконання та захисту МДР

6.1. Робота виконується в три етапи, таблиця 6.1.

Таблиця 6.1 – Етапи виконання роботи.

Етап	Зміст	Початок	Кінець	Результат
1	Інформаційний пошук та огляд літературних джерел. Розробка методу та програмної реалізації блочного порівняння файлів.			Розділи 1 та 3.
2	Техніко-економічне обґрунтування теми дипломної роботи, розробка програмного засобу.			Чернетки матеріалів 1 розділу. Попередній захист.
3	Підготовка матеріалів пояснювальної записки.			Пояснювальна записка.

7. Загальний алгоритм роботи програми є таким:

20) Запуск програми.
21) Якщо задана команда виходу, перейти до п. 19. Інакше – до п. 3.
22) Виконати архівацію. Якщо початковий файл вже завантажено, йти до пункту 14. Інакше прямуй на пункт 4.
23) Вибрати ім’я нового файлу.
24) Отримати команду користувача. Якщо потрібно додати файл в список, йди до п. 6. Якщо потрібно змінити файл, йди до п. 7.

25) Додати файл в список. Якщо потрібно видалити файл, йти до п.8.

    Якщо потрібно створити архів, йти до п.9.
26) Змінити файл.
27) Видалити файл.
28) Створити архів.
29) Вибрати файл для архіву.
30) Стиснути файл та долучити до потоку.
31) Якщо архів пустий, йти на пункт 10. Інакше йти на пункт 13.
32) Зберегти потік як архів. Перейти на пункт 2.
33) Завантажити початковий файл.
34) Вибрати файл.
35) Вилучити файл із потоку і розпакувати.
36) Зберегти файл.
37) Якщо список пустий, йти на пункт 2. Інакше пункт 15.
38) Кінець роботи програми.

8. Порядок контролю та прийому

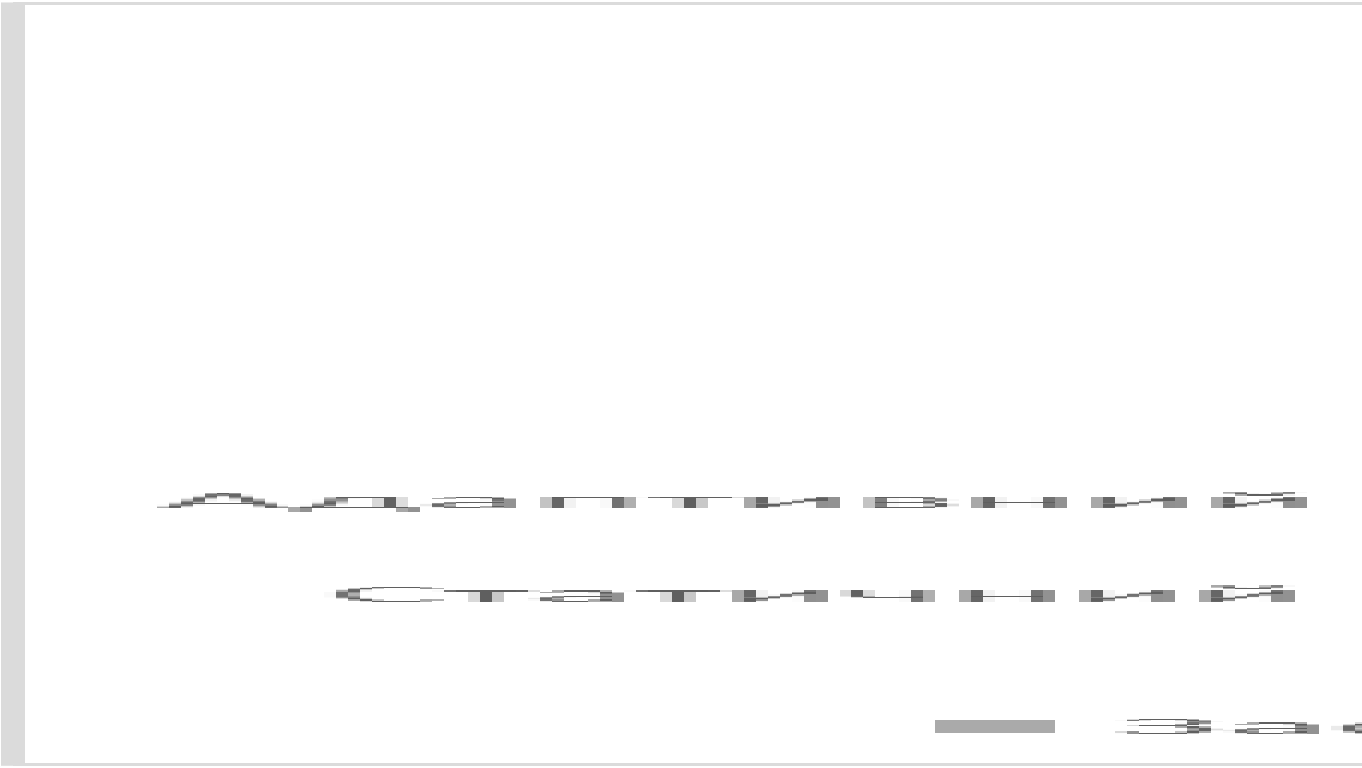
8.1 До приймання дипломної роботи надається:

- пояснювальна записка з відповідними узгодженнями;
- демонстрація програмного засобу;
- презентація;
- відгук керівника роботи та рецензента.

Технічне завдання до виконання прийняв _____ Чирва П. В.

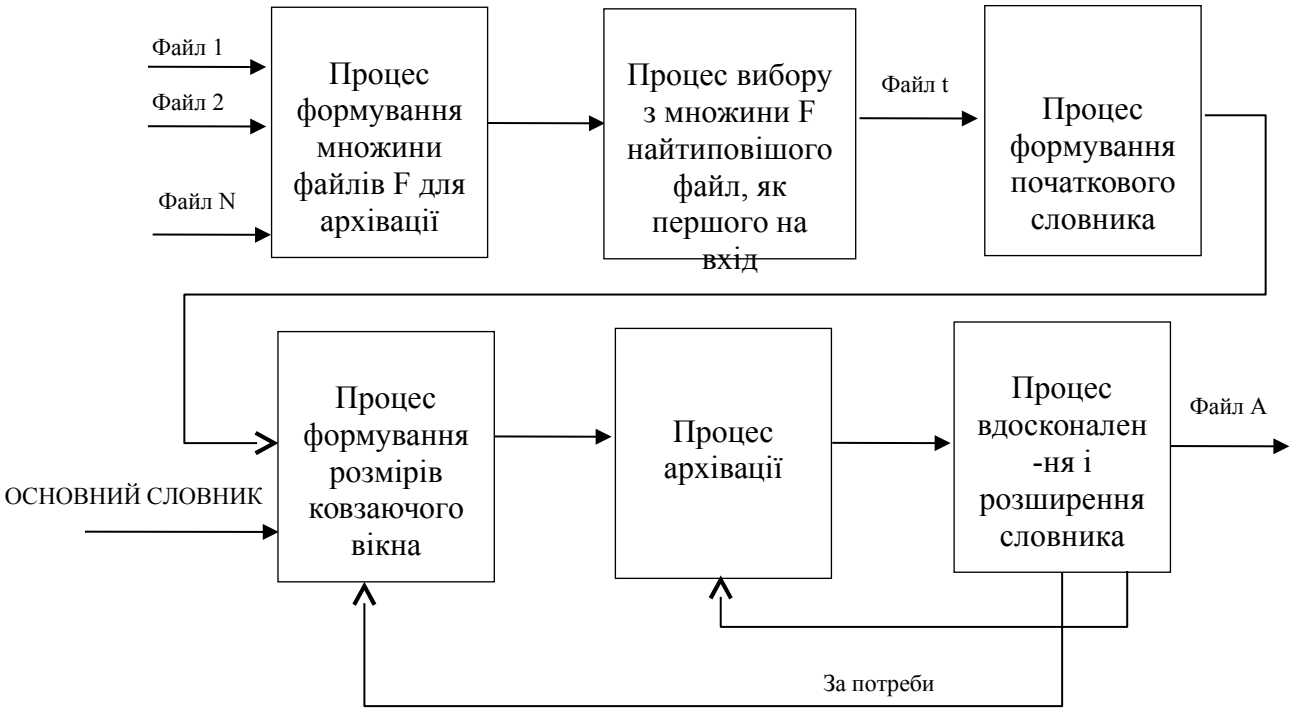
Додаток Б

Порівняння алгоримів архівації та стиснення у випадку роботи з однотипними даними



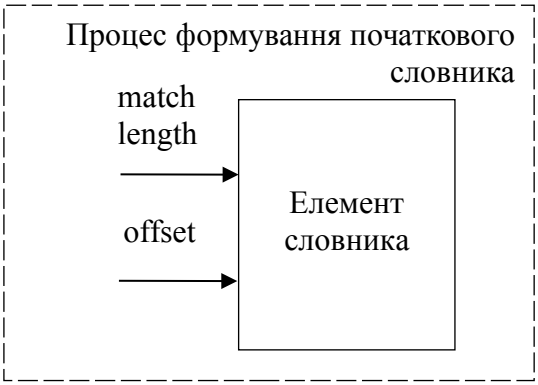
Додаток В

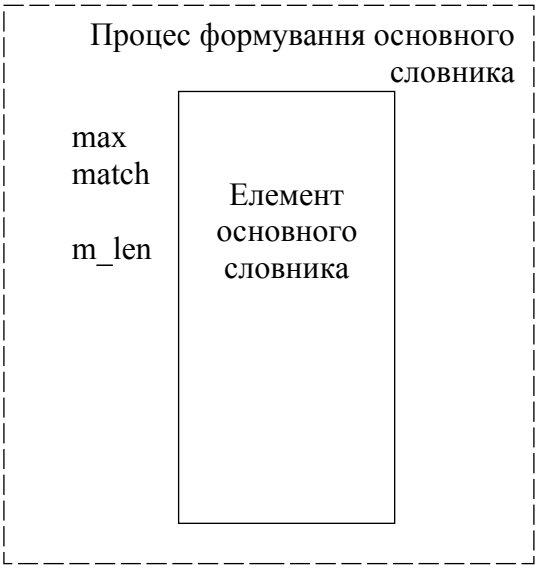
Загальна схема методу архівації однотипних файлів



Додаток Г

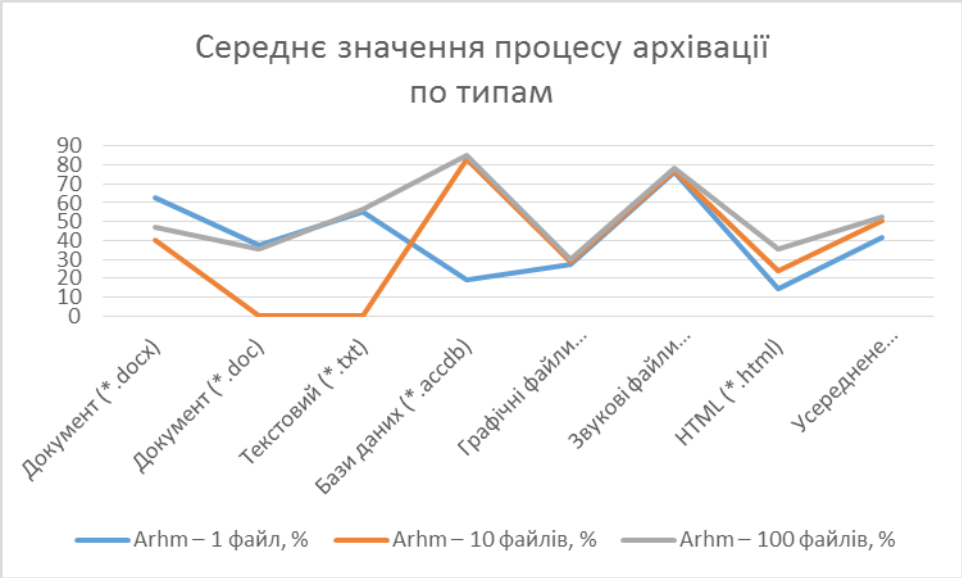
Процес формування словника

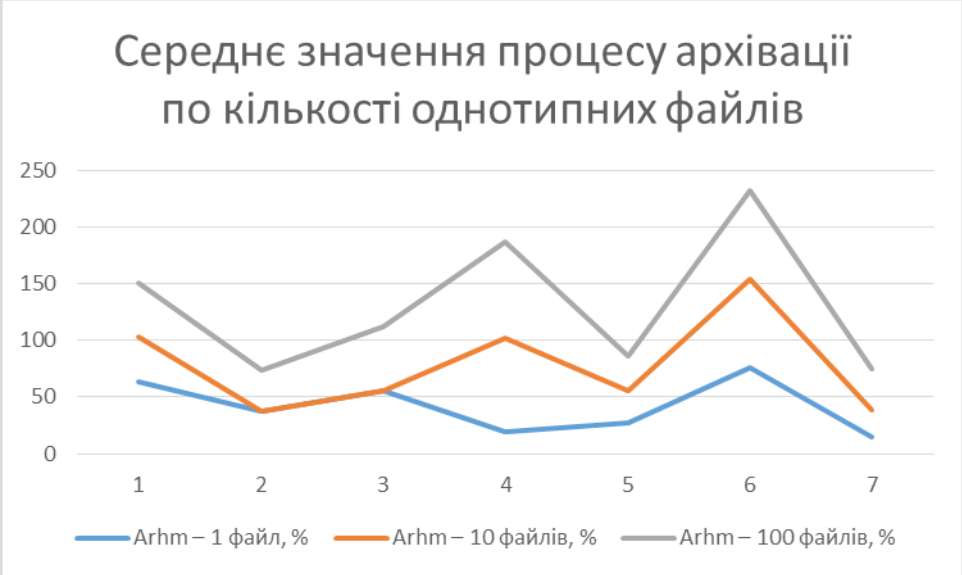




Додаток Д.

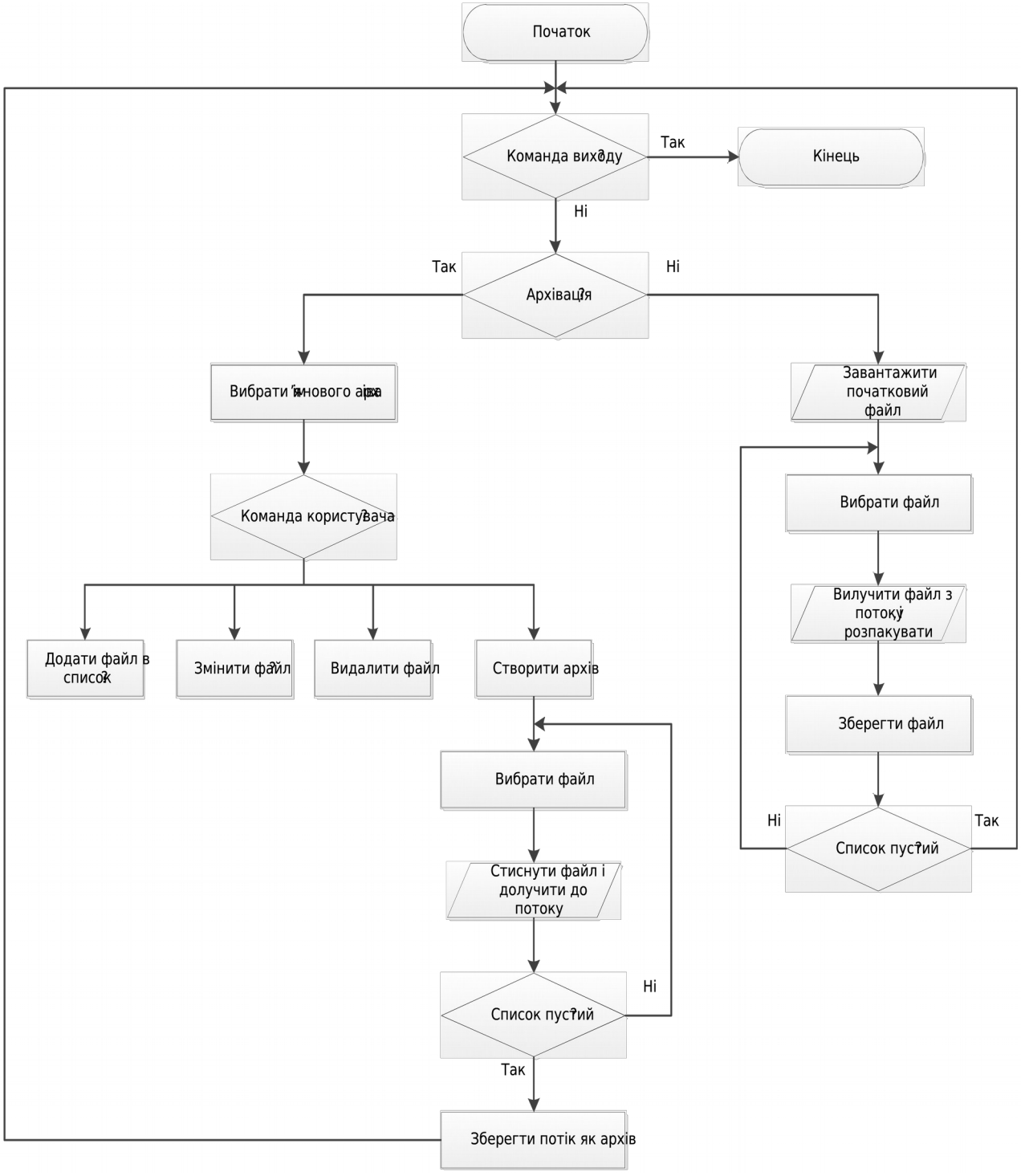
Діаграма процесу архівації





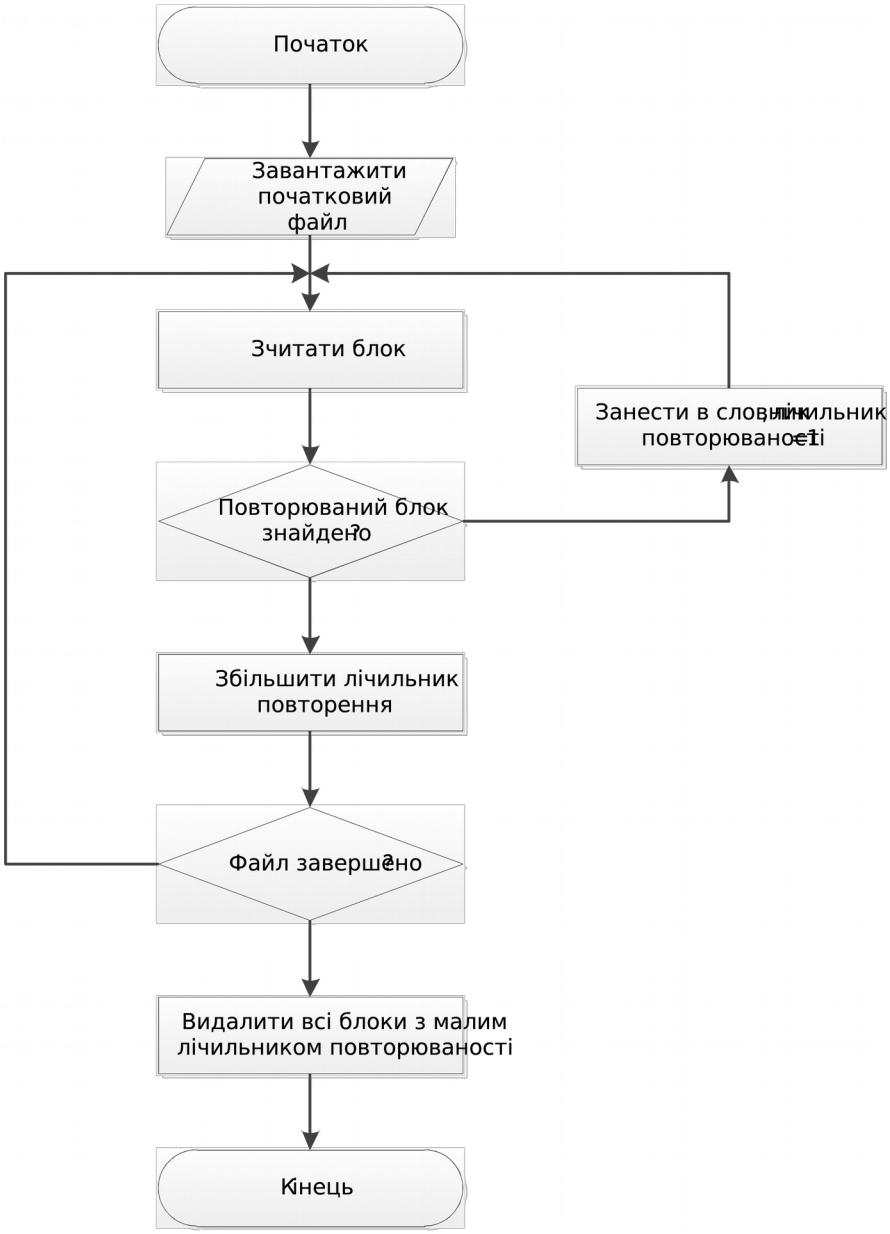
Додаток Е

Блок-схема алгоритму роботи програми



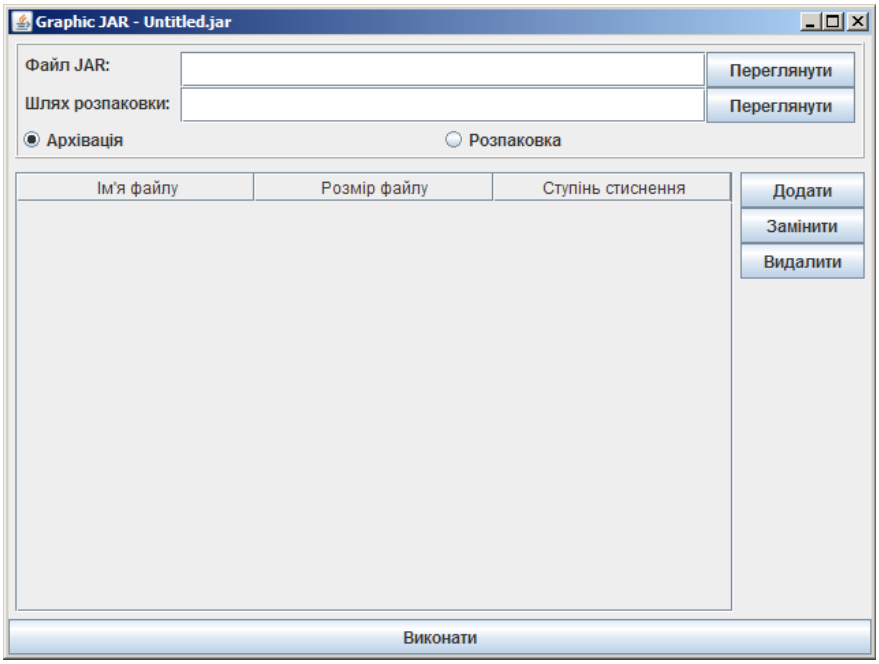
Додаток К

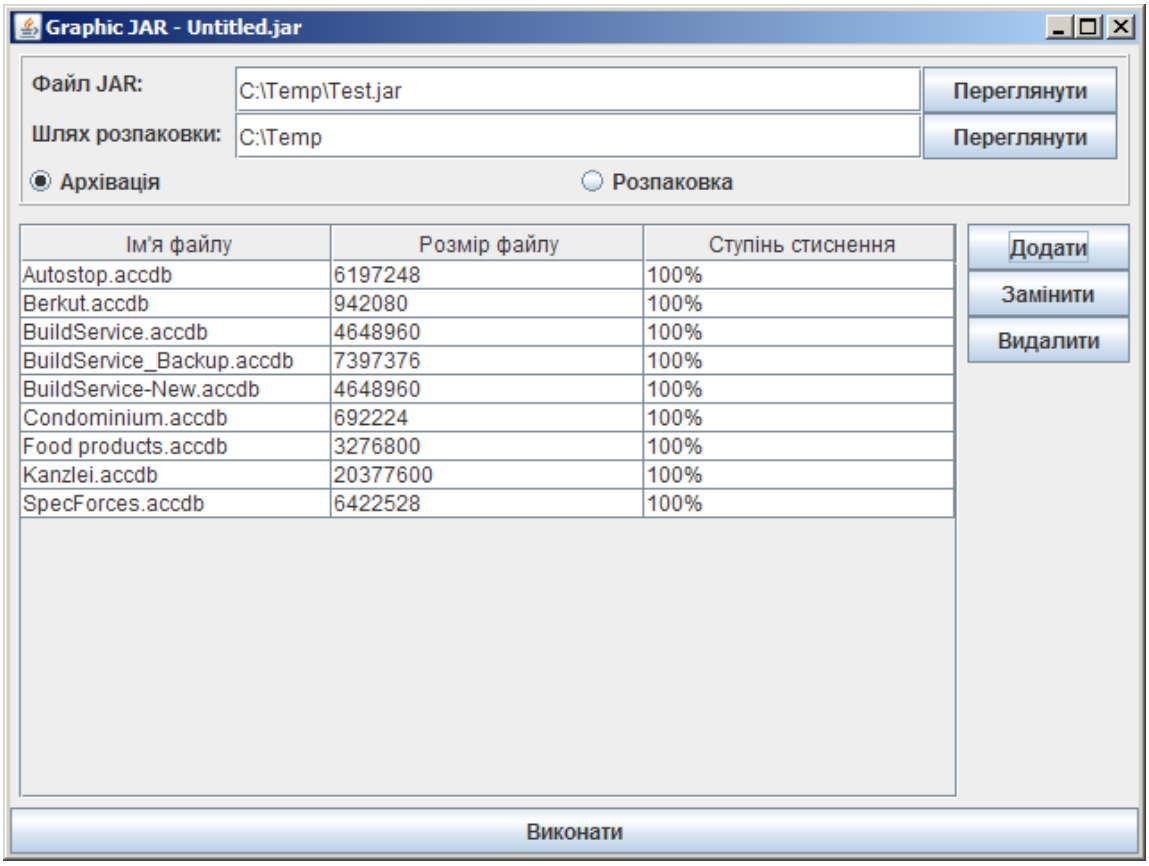
Оцінка конкурентів



Додаток Ж

Дизайн користувацького інтерфейсу





Додаток М

Лістинг програми

```
package ua.vntu.graphjar;

import java.awt.BorderLayout;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEv ent;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

import java.io.InputStream;

import java.nio.file.Files;
import java.nio.file.Path;

import java.nio.file.Paths;

import java.text.DecimalFormat;

import java.util.Enumeration;
import java.util.Vector;

import java.util.jar.JarEntry;

import java.util.jar.JarFile;
import java.util.jar.JarInputStream;
import java.util.jar.JarOutputStream;

import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
```

```
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.border.CompoundBorder;
import javax.swing.border.EmptyBorder;
import javax.swing.border.EtchedBorder;
import javax.swing.table.DefaultTableModel;

public class GraphJar extends JFrame
  implements ActionListener, ItemListener
{
  private String currentArchive = "Untitled.jar";

  private JTextField jarFile = new JTextField();
  private JTextField unpackPath = new JTextField();

  private JRadioButton flagPack = new JRadioButton("Архівація");
  private JRadioButton flagUnPack = new JRadioButton("Розпаковування");

  private JTable fileList = null;

  public class FileRecord
  {
   private File currentFile;

   public FileRecord(File curF) {currentFile = curF;}
   public String toString() {return currentFile.getName();}
  }

  public GraphJar()
  {
    super();
    setSize(640, 480);
    setTitle("Graphic JAR - " + currentArchive);

    getContentPane().setLayout(new BorderLayout());

    JPanel paths = new JPanel(new BorderLayout());
    JPanel innerPane = new JPanel(new GridLayout(2, 1));
    innerPane.add(new JLabel("Файл JAR: "));
    innerPane.add(new JLabel("Шлях розпаковування: "));
    innerPane.setBorder(new EmptyBorder(0, 5, 0, 5));
    paths.add(innerPane, BorderLayout.WEST);
```

```
    innerPane = new JPanel(new GridLayout(2, 1));
    innerPane.add(jarFile);
    innerPane.add(unpackPath);
    innerPane.setBorder(new EmptyBorder(5, 0, 0, 0));
    paths.add(innerPane, BorderLayout.CENTER);

    innerPane = new JPanel(new GridLayout(2, 1));
    innerPane.add(createButton("Переглянути", "browseFile"));
    innerPane.add(createButton("Переглянути", "browsePath"));
    innerPane.setBorder(new EmptyBorder(5, 0, 0, 5));
    paths.add(innerPane, BorderLayout.EAST);

    ButtonGroup bg = new ButtonGroup();
    bg.add(flagPack);
    bg.add(flagUnPack);
    flagUnPack.addItemListener(this);
    flagPack.setSelected(true);

    innerPane = new JPanel(new GridLayout(1, 2));
    innerPane.add(flagPack);
    innerPane.add(flagUnPack);
    paths.add(innerPane, BorderLayout.SOUTH);
       paths.setBorder(new CompoundBorder(new EmptyBorder(5, 5, 5, 5), new
EtchedBorder(EtchedBorder.RAISED)));

    getContentPane().add(paths, BorderLayout.NORTH);

    Vector header = new Vector();
    header.add("Ім’я файлу");
    header.add("Розмір файлу");
    header.add("Ступінь процесу стиснення та архівації ");

    fileList = new JTable(new Vector(), header);
    innerPane = new JPanel(new BorderLayout());
    innerPane.add(new JScrollPane(fileList), BorderLayout.CENTER);
    innerPane.setBorder(new EmptyBorder(5, 5, 5, 5));

    JPanel buttonPane = new JPanel(new BorderLayout());
    JPanel top = new JPanel(new GridLayout(3, 1));
    top.add(createButton("Додати", "addFile"));
    top.add(createButton("Замінити", "replaceFile"));
    top.add(createButton("Видалити", "removeFile"));
    buttonPane.add(top, BorderLayout.NORTH);
    buttonPane.add(new JPanel(), BorderLayout.CENTER);
    buttonPane.setBorder(new EmptyBorder(0, 5, 0, 0));
    innerPane.add(buttonPane, BorderLayout.EAST);
```

```
    getContentPane().add(innerPane, BorderLayout.CENTER);

    getContentPane().add(createButton("Виконати", "performAction"),
BorderLayout.SOUTH);

    addWindowListener(new WindowAdapter()
    {
     public void windowClosing(WindowEvent e)
     {
      System.exit(0);
     }
    });
  }

  private JButton createButton(String title, String command)
  {
   JButton button = new JButton(title);
   button.setActionCommand(command);
   button.addActionListener(this);
   return button;
  }

  private byte[] getBytesFromFile(String canonPath)
  throws IOException
  {
   Path path = Paths.get(canonPath);
   return Files.readAllBytes(path);
  }

  private void clearTable()
  {
   long tableSize = fileList.getRowCount();

   for(int i = 0; i < tableSize; i++)
    ((DefaultTableModel)fileList.getModel()).removeRow(0);
  }

  private void createJarFile()
  {
   try
   {
    FileOutputStream fos = new FileOutputStream(jarFile.getText());
    JarOutputStream jar = new JarOutputStream(fos);
    for(int i = 0; i < fileList.getRowCount(); i++)
    {
     File current = ((FileRecord)fileList.getModel().getValueAt(i, 0)).currentFile;
```

```
    JarEntry entry = new JarEntry(current.getName());
    jar.putNextEntry(entry);
    jar.write(getBytesFromFile(current.getCanonicalPath()));
    jar.closeEntry();

                long fullSize = Long.parseLong(fileList.getModel().getValueAt(i,
1).toString());
    DecimalFormat df = new DecimalFormat("####0.00");
    double percent = ((double)entry.getCompressedSize() / (double)fullSize) * 100;
    ((DefaultTableModel)fileList.getModel()).setValueAt(df.format(percent) + "%",
i, 2);

   }
   jar.close();
   fos.close();
  } catch (Exception e)
  {
   System.err.println(e.getMessage());
   e.printStackTrace(System.err);
  }
 }

 public void openJarFile(String what)
 {
  try
  {
   JarFile jar = new JarFile(what);
   Enumeration entries = jar.entries();
   while(entries.hasMoreElements())
   {
    JarEntry entry = (JarEntry) entries.nextElement();

    double percent = ((double)entry.getCompressedSize() / (double)entry.getSize())
* 100;

    Vector newRow = new Vector();
    newRow.add(entry.getName());
    newRow.add(entry.getSize());
    DecimalFormat df = new DecimalFormat("####0.00");
    newRow.add(df.format(percent) + "%");

    ((DefaultTableModel)fileList.getModel()).addRow(newRow);
   }
   jar.close();
  } catch (Exception e)
  {
   System.err.println(e.getMessage());
```

```
    e.printStackTrace(System.err);
   }
  }

  public void extractFromJar(String what, String name)
  {
   try
   {
    JarFile jar = new JarFile(what);
    Enumeration entries = jar.entries();
    while(entries.hasMoreElements())
    {
     JarEntry entry = (JarEntry) entries.nextElement();

     if(entry.getName().equals(name))
     {
          FileOutputStream fos = new FileOutputStream(unpackPath.getText() +
File.separator + entry.getName());
      InputStream is = jar.getInputStream(entry);
      while(is.available() > 0)
       fos.write(is.read());
      fos.close();
      is.close();
     }
    }
    jar.close();
   } catch (Exception e)
   {
    System.err.println(e.getMessage());
    e.printStackTrace(System.err);
   }
  }

  public void actionPerformed(ActionEvent e)
  {
   if(e.getActionCommand().equals("browseFile"))
   {
    JFileChooser chooser = new JFileChooser();
    chooser.setFileFilter(new JarFileFilter());
    boolean accepted = false;
    if(flagPack.isSelected())
    {
     if(chooser.showSaveDialog(this) == JFileChooser.APPROVE_OPTION)
      accepted = true;
    }
    else
    {
```

```
    if(chooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION)
     accepted = true;
    }
       try
    {
    if(accepted)
    {
     jarFile.setText(chooser.getSelectedFile().getCanonicalPath());
     unpackPath.setText(chooser.getSelectedFile().getParent());
          if(flagUnPack.isSelected())
      openJarFile(chooser.getSelectedFile().getCanonicalPath());
    }
    } catch (IOException exc)
    {
    System.err.println(exc.getMessage());
    exc.printStackTrace(System.err);
    }
   }
  if(e.getActionCommand().equals("addFile"))
   {
   JFileChooser chooser = new JFileChooser();
   if(chooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION)
   {
   Vector newRow = new Vector();
   newRow.add(new FileRecord(chooser.getSelectedFile()));
   newRow.add(String.valueOf(chooser.getSelectedFile().length()));
   newRow.add("100%");
   ((DefaultTableModel)fileList.getModel()).addRow(newRow);
   }
   }
  if(e.getActionCommand().equals("replaceFile"))
   {
   JFileChooser chooser = new JFileChooser();
   if(chooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION)
   {
    int currentRow = fileList.getSelectedRow();
                              ((DefaultTableModel)fileList.getModel()).setValueAt(new
FileRecord(chooser.getSelectedFile()), currentRow, 0);
     ((DefaultTableModel)fileList.getModel()).setValueAt(String.valueOf(chooser.g
etSelectedFile().length()), currentRow, 1);
     ((DefaultTableModel)fileList.getModel()).setValueAt("100%", currentRow, 2);
    }
   }
   if(e.getActionCommand().equals("removeFile"))
   {
  ((DefaultTableModel)fileList.getModel()).removeRow(fileList.getSelectedRow())
;
```

```
    }
        if(e.getActionCommand().equals("performAction"))
    {
     if(!jarFile.getText().equals(""))
     {
      if(flagPack.isSelected())
      {
       createJarFile();
       JOptionPane.showConfirmDialog(this, "Архівація завершена");
      }
      else
      {
                                                    extractFromJar(jarFile.getText(),
fileList.getModel().getValueAt(fileList.getSelectedRow(), 0).toString());
       JOptionPane.showConfirmDialog(this, "Розпаковування завершена");
      }
     }
    }
  }
  public void itemStateChanged(ItemEvent e)
  {
   if(flagUnPack.isSelected())
    clearTable();
  }
  public static void main(String[] args)
  {
    GraphJar graphJar = new GraphJar();
    graphJar.setVisible(true);
  }
}
package ua.vntu.graphjar;
import java.io.File;
import javax.swing.filechooser.FileFilter;
public class JarFileFilter extends FileFilter
{
  public JarFileFilter()
  {
    super();
  }
  public boolean accept(File f)
  {
    if(f.getName().endsWith(".jar") || f.isDirectory())
     return true;
    return false;
  }

  @Override
```

```
  public String getDescription()
  {
    return "(*.JAR) JAR Archive";
  }
}
```

Лістинг програми JarFileFilter.java

```
package ua.vntu.graphjar;
import java.io.File;
import javax.swing.filechooser.FileFilter;
public class JarFileFilter extends FileFilter
{
  public JarFileFilter()
  {
    super();
  }
  public boolean accept(File f)
  {
    if(f.getName().endsWith(".jar") || f.isDirectory())
     return true;
    return false;
  }
  @Override
  public String getDescription()
  {
    return "(*.JAR) JAR Archive";
  }
}
```