

РЕФЕРАТ

Пояснювальна записка до курсової роботи складається з 27 сторінок, 3 джерел, 8 рисунків, 2 таблиць

Предмет дослідження: алгоритм стиснення зображень PNG

Мета курсової роботи: систематизація, поглиблення та активне використання знань з теорії інформації та кодування, основ програмування, отриманих підчас вивчення лекційного курсу, а також на практичних та лабораторних заняттях на прикладі вирішення задач стиснення файлів.

Методи дослідження: вивчення літературних джерел, створення та відлагодження програм на комп'ютері, аналіз результатів роботи програми.

В роботі розроблена програма для стиснення зображень у форматі PNG. Створений продукт дозволяє перетворити вхідні зображення на вихідні зображення у форматі PNG.

КЛЮЧОВІ СЛОВА: АЛГОРИТМ, СТИСНЕННЯ, PNG, DEFLATE, LZ-77, ХАФФМАН.

ЗМІСТ

ВВЕДЕННЯ	5
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ	6
1.1 Аналіз технічного завдання.....	6
1.2 Опис структурної моделі програми.....	7
1.3 Опис функціональної моделі програми	8
2 ОПИС АЛГОРИТМУ СТИСНЕННЯ ТА ФОРМАТУ PNG.....	10
2.1 Стиснення	10
2.2 Структура формату	16
3 РЕАЛІЗАЦІЯ ПРОГРАМИ ТА ЇЇ ТЕСТУВАННЯ.....	21
3.1 Вибір середовища розробки	21
3.2 Бібліотеки, що використані при розробці програми	21
3.3 Функції коду програми	23
3.4 Тестування програми	25
ВИСНОВКИ.....	26
СПИСОК ДЖЕРЕЛ.....	27
ДОДАТОК.....	28

ВВЕДЕННЯ

Якщо зазирнути до історії, то можна прослідкувати, як з моменту появи перших електронних обчислювальних машин розробники намагаються урізноманітнити способи спілкування користувача та машини. Це спілкування було б більш обмеженим, якби не використовувало один з найбільш простих способів — мова зображень та образів. Сьогодні графічні зображення на екрані монітору стали для нас нормою, цілком невід’ємним атрибутом інтерфейсу. Спектр використання комп’ютерної графіки, окрім засобів інтерфейсу, надзвичайно широкий: від створення рекламних роликів, комп’ютерних фільмів та ігор, моделей одєжі, архітектурних споруд, комп’ютерного живопису до візуалізації результатів наукових досліджень. Можна впевнено сказати, що популярність Internet та, зокрема, WWW, багато в чому пояснюється широким використанням графіки. Одним з основних параметрів графічних файлів є розмір файлу, та в переважній більшості випадків доцільно його зменшити, причому щоб сам файл лишався незмінним або результуючі зміни були неістотними. Для цього існують різні алгоритми стиснення, зокрема стиснення формату PNG.

Основною задачею цієї курсової роботи є дослідження способів стиснення графічних файлів в цілому, стиснення PNG зокрема та порівняння його з іншими форматами, виокремити переваги та недоліки використання формату PNG, а також реалізувати програму, що реалізує таке стиснення та тестує його на різних графічних файлах. Ця програма також має виводити результати цього стиснення (розмір результуючого файлу).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз технічного завдання

Стиснення (англ. compression) – це процес перекодування даних з метою зменшення їх розміру (об'єму) та базується на усуненні надлишку інформації (зазвичай – повторювані послідовності замінюються коротшим значенням). При цьому, стиснення даних що не мають надлишку, наприклад випадковий сигнал або шум, неможливе.

Розрізняють два види стиснення інформації:

- Стиснення з втратами (англ. lossy compression) – метод стиснення даних, при застосуванні якого, відновленні дані відрізняються від вихідних, але ця різниця не є істотною з точки зору подальшого використання цих даних.
- Стиснення без втрат (англ. lossless compression) – метод стиснення даних, при застосуванні якого, скомпресовані дані можна однозначно відновити з точністю до байта, пікселя, тощо.

Розроблена програма призначена для стиснення зображень у формат PNG, порівняння розміру отриманих PNG-зображень з нестисненими (як приклад, взято формат BMP) а також з іншими форматами – JPEG та GIF. Продукт дозволяє обрати папку у якій знаходяться зображення та відображає результати стиснення: розміри вихідних та скомпресованих зображень, що зберігаються до папки, що створюється всередині обраної.

Розроблений продукт виконуватиме ряд функцій, що оголошені в технічному завданні, а саме:

- Організація обрання папки, в якій містяться зображення, які треба скомпресувати.
- Контроль отриманих даних – якщо в папці містяться неграфічні файли, вони ігноруються
- Організація стиснення отриманих зображень
- Організація виведення стиснених файлів – їх збереження до окремої папки

1.2 Опис структурної моделі програми

Структурна модель програми представлена на рис. 1.1. Структурна схема включає до себе три основні підсистеми: підсистема введення, підсистема компресування, підсистема виведення.

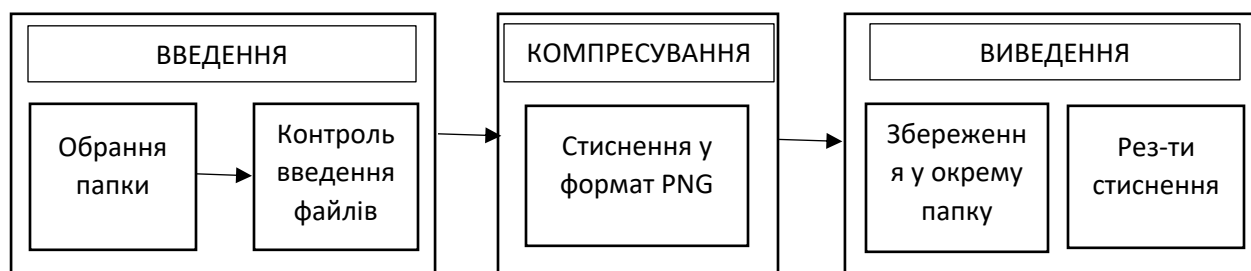


рис. 1.1 – структурна схема програми

Підсистема введення файлів відповідає за отримання програмою зображень, що необхідно скомпресувати. Програма виводить діалогове вікно з файловим менеджером та пропонує обрати папку з якої отримати зображення (якщо обрати зображення, програма зчитає тільки його, причому при обранні невірному файлу, наприклад, .txt або .exe – програма видасть помилку).

Підсистема компресування відповідає за стиснення форматом PNG отриманих зображень.

Підсистема виведення відповідає за збереження файлів у окрему папку всередині обраної у підсистемі введення, а у вікні програми виводить таблицю, що показує розміри отриманих та зкомпресованих файлів та ефективність стиснення. Кожна підсистема починає роботу після завершення попередньої.

1.3 Опис функціональної моделі програми

Функціональна модель програми зображена на рис. 1.2 та відображає які функції виконує та чи інша частина програми.



рис. 1.2 – функціональна схема програми

Програма відкриває вікно з порожньою таблицею та діалогове вікно вибору папки поверх нього. Програма, отримавши шлях до папки перевіряє чи є у папці графічні файли та за їх наявності зчитує їх. У зворотньому випадку програма видає помилку про ненааявність у папці зображень. Якщо користувач обрав файл, то програма перевіряє на те чи є він графічним. Якщо ні – програма видає відповідну помилку. Далі програма виконує стиснення (алгоритм цього стиснення буде описаний далі) та врешті решт – зберігає файли у папку «compressed», що автоматично створюється у вказаній раніше папці з зображеннями (при виборі одного файлу – в тій же папці, в якій був цей файл). При цьому в таблиці вказуються усі отримані зображення, їх початкові та кінцеві розміри та ефективність стиснення.

2 ОПИС АЛГОРИТМУ СТИСНЕННЯ ТА ФОРМАТУ PNG

2.1 Стиснення

Алгоритм стиснення PNG складається з двох етапів – фільтрування та алгоритму DEFLATE. Під час фільтрування файл проходить через алгоритм дельта-кодування. Цей алгоритм дозволяє замість самих даних підставляти різниці (дельти) послідовних даних. Це дозволяє при невеликій різниці між сусідніми значеннями отримати групу набагато менших значень, які будуть часто повторюватися, що зробить подальше стиснення значно ефективнішим. У форматі PNG дельта-кодування виконується построково та кожний наступний піксель закодовується відносно сусідніх пікселів ліворуч, зверху та зверху-ліворуч від нього. На малюнку нижче ці пікселі позначені як А, В та С відповідно. Алгоритм фільтрування бере ці значення та використовує їх для «передбачення» пікселя Х. Результуюче значення Х є різницею між цим «передбаченням» та дійсним значенням Х.

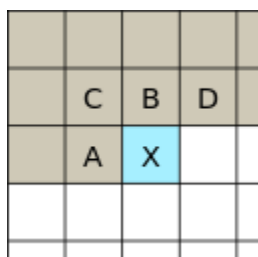


Рис. 2.1 – схематичне зображення пікселів

Оскільки значення пікселів уздовж кожної строки можуть змінюватися по-різному, то існує 5 типів фільтрування пікселів для кожного ряду:

- 0: Без фільтрації;

- 1: Різниця між поточним та лівим (X та A);
- 2: Різниця між поточним та верхнім (X та B);
- 3: Різниця між поточним та середнім арифметичним верхнього та лівого (X та $(A+B)/2$);
- 4: Paeth-предиктор – A, B або C: найближче до $P = A+B-C$ з цих значень. У випадку рівності деяких з них, найбільший пріоритет має A(ліворуч), далі B(зверху) та C(зверху-ліворуч).

Перед кожною строкою ставиться байт, що відповідає типу фільтрування застосованому на цій строці. Рисунок нижче практично показує кожен з цих типів. Слід зазначити, що ця фільтрація застосовується для значень кольорових каналів (інакше кажучи, байтів інформації), а не кольору. Тобто фільтрація застосовується послідовно спочатку для значень усіх каналів червоного кольору, далі зеленого, синього та альфа-каналу (якщо такий є у даному зображенні) – але для усіх каналів одного ряду застосовується однаковий тип фільтрування.

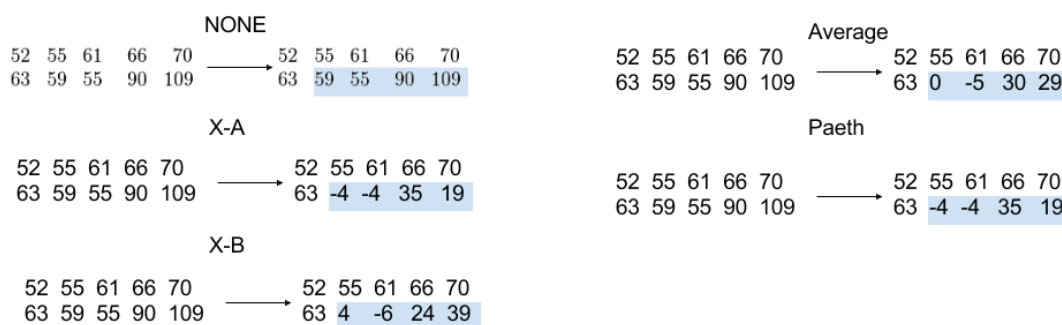


рис. 2.2 – приклади використання кожного типу фільтрування

Також PNG має певні ознаки для застосування того чи іншого типу. Для індексованих та 8-бітних кольорів застосовується 0-фільтр, а для true-color-кольорів використовується наступний метод: обчислюються значення для

кожного фільтра, беруться абсолютні значення різниць, ці значення сумуються та обирається фільтр, що має найменшу таку суму.

Стиснення за алгоритмом DEFLATE поєднує у собі алгоритм LZ77 (Lempel-Ziv) похідним від якого він є з алгоритмом Хафмана. Для кращого розуміння DEFLATE слід спершу описати його частини.

LZ77 – алгоритм, який працює відслідковуючи послідовності символів, що повторюються, та зберігає запис попередніх даних, який він «тягне за собою». Робиться це використовуючи метод, так званого, ковзаючого вікна, яке тут має розмір не більше 32 кб. Мається на увазі те, що у будь-якій точці у послідовності символів, компресор містить запис попередніх 32768 ($32 \cdot 1024$) символів (може бути менше – степенем двійки не менше 8, та не більше 15). Якщо при скануванні даних компресор знаходить серію символів, що повністю співпадає з деякою серією всередині вікна, то він замінює цю серію на два числа – перше означає відстань, тобто як далеко позаду знаходиться така ж серія бітів, друге – розмір, тобто скільки бітів повторюється. Відображемо це на прикладі нижче (риска «|» показує поточний символ, що знаходиться перед нею):

Blah blah blah blah blah!

Потік даних отримує символи «B», «l», «a», «h», « », «b», а далі іде серія символів що повторюється:

Blah b|lah blah blah!

Послідовність «lah b» після курсору повторюється з такою самою послідовністю, що знаходиться у вікні позаду курсору. Таким чином ця серія замінюється двома числами, тут – 5; 5.

Зчитані дані: Blah blah b

Їх стиснений варіант: Blah b[D: 5, L: 5]

Таке стиснення можна покращити, але тут необхідна певна «розумність»

компрессора. Наприклад, ми повернемося до двох серій, що були прийняті як однакові, та подивимося на символи після них – побачимо, що в обох випадках це «l». Тобто вже можна прийняти довжину за 6, а не 5. При подальшій перевірці ми виявлятимемо більше та більше однакових символів, навіть якщо «попередня» серія перехрещується з поточною.

Врешті решт виявиться, що 18 символів починаючи з другого співпадають з 18 символами починаючи з сьомого:

V'lah blah blah blah' blah!

Blah b'lah blah blah blah'!

Дійсно, під час відновлення, зчитуючи відстань та довжину повторюваних серій, декомпресор ще не знатиме що за символи будуть повторюватися (це викликано тим, що серії, що повторюються, накладаються), але підставляючи вже відомі символи, можна буде підставити попередню серію знову і знову, доки не відновиться вихідна послідовність (тире позначають незаповнені символи):

V'lah b-----'! → Blah b'lah b-----'!

V'lah blah b-----'! → Blah b'lah blah b-----'!

V'lah blah blah b---'! → Blah b'lah blah blah b---'!

V'lah blah blah blah' b---! → Blah b'lah blah blah blah'!

Таким чином вихідну послідовність можна зтиснути таким чином:

Blah b[D: 5, L: 18]!

з можливістю повного відновлення почтакових даних.

Алгоритм Хафмана – алгоритм префіксного кодування який будує елементи афлавіту поєднуючи їх у вигляді бінарного дерева, що створюється залежно від частоти появи символів. Така частота може бути як передана дереву заздалегідь (наприклад, значення частот появ букв, що використовуються при частотному аналізі) так і обчислена окремо для даних,

що шифруються. В якості прикладу, візьмемо алфавіт з частотами літер A(0.3), B(0.3), C(0.2), D(0.11), E(0.09). Спочатку створюється нода з найменшими значеннями, а віткам цієї ноди присвоюються значення 0 та 1 (як правило, 0 присвоюється вітці, що відповідає символу з меншою частотою, в цьому прикладі вітці D присвоюється 1, а E – 0) далі ця нода поєднується з іншим символом, частота якого найближча до суми частот символів цієї вітки, тут нода D-E поєднується з символом C, що в результаті створює ще більшу ноду з частотою 0.4. Далі створюється ще одна нода з символами A та B, та поєднується з нодою C-D-E (рис. 2.3).

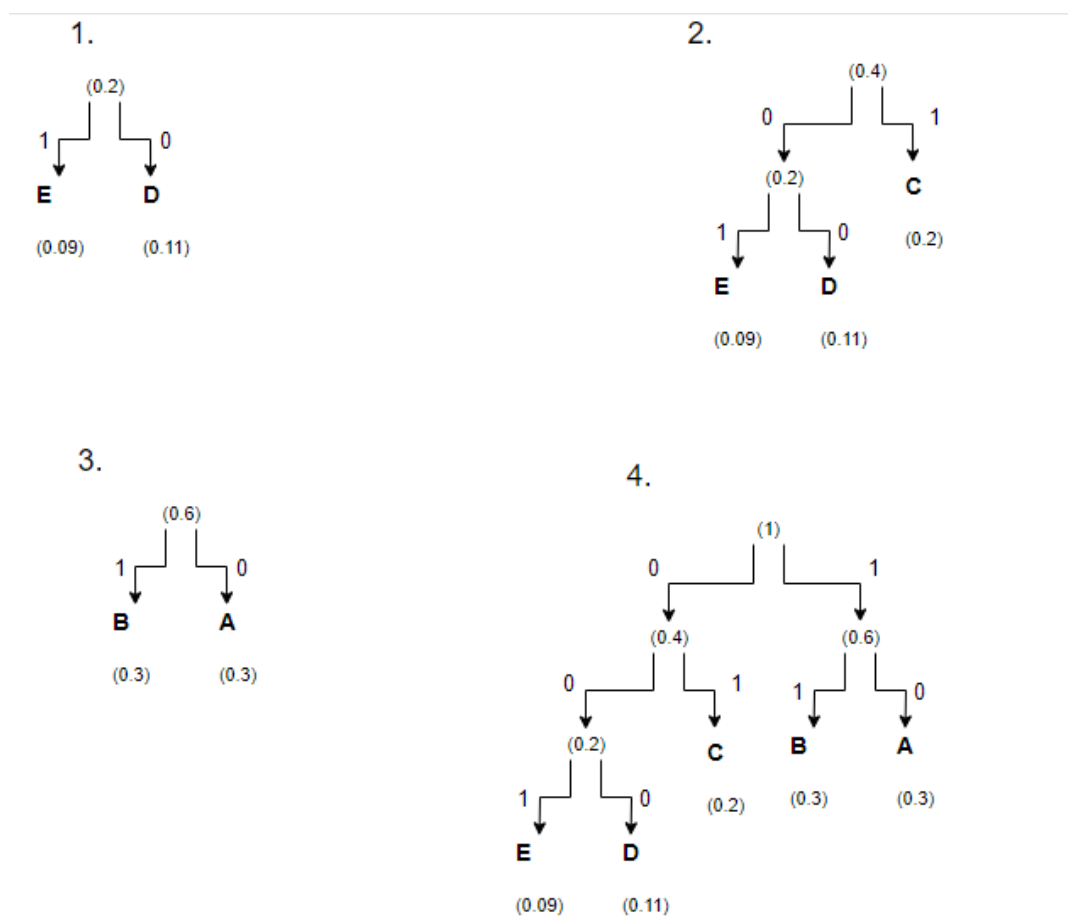


рис. 2.3 – схема створення дерева Хафмана

Таким чином символи мають такі коди:

A – 11, B – 10, C – 01, D – 001, E – 000

Структура дерева разом з даними передається наступним чином: у варіанті алгоритма Хафмана для DEFLATE є два додаткових правила. Перше: елементи з коротшими кодами ставляться ліворуч від елементів з довгими кодами. В даному прикладі D та E мають найдовші коди, тому вони ставляться праворуч. Друге: серед елементів з кодами однакової довжини, ті, що йдуть першими у наборі елементів ставляться ліворуч – тут D та E мають однакові довжини кодів, тож D отримує нульову вітку, бо йде в алфавіті перед E.

Сам алгоритм DEFLATE є доволі гнучким стосовно того який метод обирати. Усього таких методів три:

- Без стиснення. Такий метод, наприклад, застосовується для даних що вже були стиснені.
- Стиснення LZ77 а далі кодування Хафмана. Дерево Хафмана визначається специфікацією DEFLATE та не потребує додаткового місця у блоці для збереження.
- Стиснення LZ77 а далі кодування Хафмана. Дерево Хафмана генерується для даного блоку та зберігається разом з ним.

Після того, як строка пройшла фільтрування, вона передається до DEFLATE. Цей потік розбивається на блоки, кожен з яких має свій метод стиснення DEFLATE.

2.2 Структура формату

PNG файл починається з 8-байтового заголовку: «89 50 4E 47 0D 0A 1A 0A»

Цей заголовок (його ще називають підписом) ідентифікує файл як PNG та передбачає звичні проблеми передачі файлів. Перші два байти визначають PNG файл на системах, що очікують ідентифікації файлу цими байтами. Перший байт є не-ASCII значенням, що зменшує вірогідність того що текстовий файл помилково визначиться як PNG. Наступні три байти є назвою формату (P N G у ASCII). Далі йде CR-LF (carriage return та line feed послідовність що передбачає неправильні передачі файлів зі зміною послідовностей нових строк. Сьомий байт зупиняє відображення файлу у MS-DOS. Останній перехід на нову строку перевіряє виправлення проблеми з CR-LF.

Далі іде серія блоків, що починається IHDR блоком та закінчується IEND блоком. IHDR містить у собі розмір зображення у пікселях (8 байт, ширина та висота по 4 байти – цілі числа від 1 до 2^{31}), та 5 параметрів:

- Бітова глибина – однобайтове ціле, що показує кількість бітів для екземпляру кольору або індексу кольору. Дійсні значення: 1, 2, 4, 8, 16 та не всі з них допустимі для кожного типу кольору.
- Тип кольору: однобайтове ціле, що описує інтерпретацію даних зображення. Кожен з них є сумою наступних значень: 1 – використовується палітра, 2 – використовується колір, 4 – використовується альфа-канал:

табл. 2.2.1 – типи кольору

Значення	Допустимі значення БГ	Інтерпретація
0	1, 2, 4, 8, 16	Кожен колір – відтінок сірого
2	8, 16	Кожен колір є RGB значенням
3	1, 2, 4, 8	Кожен колір – індекс палітри, для цього має бути PLTE блок
4	8, 16	Кожен колір є відтінком сірого з рівнем прозорості
6	8, 16	Кожен колір є RGB значенням з рівнем прозорості

- Метод стиснення є однобайтним цілим що ідентифікує метод стиснення. Наразі існує лише 0-метод (описаний вище). Створений на випадок майбутнього розширення формату. Будь-яке ненульове значення зараз ідентифікується як помилка.
- Метод фільтру є однобайтним цілим, що ідентифікує метод фільтрування зображення (не плутати з типами фільтрування окремих рядів). Як і з методом стиснення – зараз існує лише один метод фільтрування, що позначається нульовим значенням.
- Метод переплетення є однобайтним цілим що визначає порядок передачі даних зображення. Зараз існує два можливих значення – 0 (без переплетення) та 1 (переплетення Adam7). Алгоритм Adam7 – це алгоритм черезстрокової передачі зображення, запропонований Адамом Костелло. Алгоритм передбачає розбиття зображення на 7 підзображень, підстановкою таких блоків 8*8 пікселів по усьому зображенню:

табл. 2.2.2 – таблиця методу Adam7

1	6	4	6	2	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7
3	6	4	6	3	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7

Де числа означають порядковий номер підзображення.

Блок PLTE містить від 1 до 256 елементів, кожен є трьохбайтовим значенням кольору (RGB). Довжина блоку визначає розмір палітри, будь-яке значення розміру, що не ділиться на 3 є помилкою.

Блок IDAT містить дані отримані від алгоритму стиснення. Таких блоків може бути декілька та їх набір (або сам IDAT блок якщо він один) є потоком даних формату zlib, який створює DEFLATE.

Блок IEND не містить у собі даних та позначає кінець файлу.

Кожен з цих блоків також містить чотирибайтове поле типу (їх значення відповідають назвам блоків).

Окрім цих блоків у файлі також можуть бути присутні додаткові блоки:

- bKGD – визначає колір фону на якому представляється зображення. Для типу кольору 3 (індексований колір) використовується 1 байт, для 0 та 4 (відтінки сірого з та без альфа-каналу) 2 байти, для 2 та 6 (RGB з та без альфа-каналу) – 6 байт.
- sHRM – задає кольоровий простір за CIE 1931. Містить 32 байти.

- gAMA – задає гаму зображення. Є 4-байтовим цілим, що означає значення гамми помножене на 100 000. Наприклад, значення гами 0.45 запишеться як 45000
- hIST – існує тільки за наявності палітри – показує частоту появи кожного кольору з неї. Містить послідовність двохбайтових беззнакових цілих. Якщо цей блок є, то він знаходиться після блоку палітри та перед першим IDAT.
- pHYs – відображає розмір пікселя (кількість пікселів за одиницю довжини). Містить два чотирибайтових значення Xratio та Yratio та однобайтовий ідентифікатор одиниці довжини. Якщо останній дорівнює 0, одиниця довжини вважається невідомою, тоді - як Xratio та Yratio беруться просто розмір зображення, розмір самих пікселів лигається невідомим. Якщо ідентифікатор дорівнює одиниці, то за одиницю довжини береться метр.
- sBIT – (significant bits) визначає «кольорову точність» зображення для більш простого декодування. Для типу кольору 0 – 1 байт, для 2 та 3 – три байти, для 4 – два байти, для 6 – чотири байти.
- tEXt – текстові дані які можна зберегти разом із зображенням. Ключові слова можуть мати розмір від 2 до 80 байт (з нульовим символом) та сам текст – довільний розмір. До ключових слів входять назва, автор, опис зображення, нотатка про авторські права, час створення, програмне забезпечення за допомогою якого зображення було створено, дисклеймер, попередження про вміст, джерело та коментарій.
- tIME – показує дату та час останньої модифікації файлу: рік, місяць, день, час, мінути, секунди: для року два байти, для інших – один, разом: 7 байт
- tRNS – містить інформацію про прозорість. Для типу кольору 3 міститься ряд однобайтових значень для кожного кольору палітри. Для типу 0

мається одне двобайтове значення сірого кольору. Для типу 2 міститься одне RGB значення розміром 6 байт. Для типів 4 та 6 цей блок не використовується оскільки в них вже є альфа-канал.

- zTXt – стиснені текстові дані. Абсолютно ідентичний блоку tEXt, але текстові дані ще стискаються. Корисний для збереження великих текстів.

3 РЕАЛІЗАЦІЯ ПРОГРАМИ ТА ЇЇ ТЕСТУВАННЯ

3.1 Вибір середовища розробки

В якості мови програмування я обрав Java, бо вона є об'єктно-орієнтованою, мультиплатформенною, має зручні способи успадковування та має безпечну типізацію. Як платформу для створення віконних програм я обрав Java FX, яка, дозволяє зручно створювати на Java віконне програмне забезпечення, а Scene Builder дозволяє оформлювати вікно не кодом а візуально, за допомогою WYSIWIG інтерфейсу.

В якості середовища розробки я обрав IntelliJ IDEA компанії JetBrains через його великий функціонал, перевірку та відладку коду під час його написання та можливість обирати методи та поля із зручного меню.

3.2 Бібліотеки що використані при розробці програми

- `java.imageio` – бібліотека, що дозволяє виконувати зручне зчитування та запис зображень та виконувати просте кодування та декодування зображень.
- `java.awt` – бібліотека, що містить класи для створення інтерфейсу користувача та для роботи з графікою та зображеннями.
- `java.io` – бібліотека, що містить класи для роботи з введенням та виведенням даних через потоки та файлову систему.
- `javafx.*` - бібліотека, що дозволяє створювати програми типу Rich Internet Application

3.3 Функції коду програми

Програма містить 5 класів та 1 .fxml файл. У файлі sample.fxml міститься структура вікна програми.

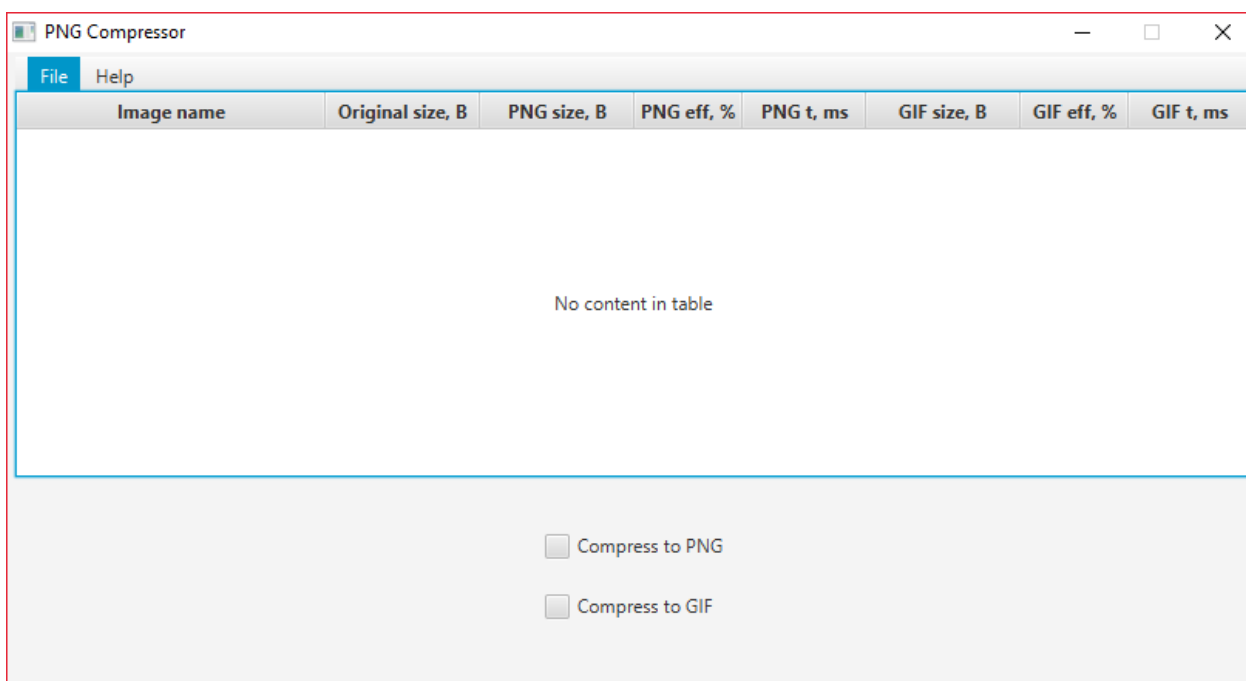


рис. 3.1 – вікно програми

Клас Main відповідає за запуск програми та ініціалізує її віконний інтерфейс.

Клас Controller містить у собі функції, що виконуються при роботі у інтерфейсі (натискання кнопки, вибір прапорця тощо).

- `getStage()` – отримує об'єкт типу `Stage` з класу `Main` для подальшого використання для подальшої ініціалізації діалогових вікон вибору файлу.
- `handleCompression()` – викликає діалогове вікно вибору файлу, після вибору якого виконує стиснення (якщо обраний прапорець «Compress to PNG» - виконує стиснення у формат PNG, якщо «Compress to GIF» - у

GIF, можуть бути обрані обидва та жодний з них) та зберігає результуючі *.png та *.gif файли у ту саму папку.

- `handleAbout()` – виводить діалогове вікно з інформацією про розробників.

Клас `tableData` містить у собі дані для занесення до таблиці (назва вихідного файлу, його розмір, розмір стисненого файлу, ефективність та час стиснення для PNG та GIF).

Клас `Compression` містить функцію для власне стиснення зображення. Його метод `compress` приймає два булевих значення, що відповідають прапорцям «Compress to PNG» та «Compress to GIF».

Клас `Dialogs` містить методи для ініціалізації діалогових вікон – вікно вибору файлу та вікно `About`.

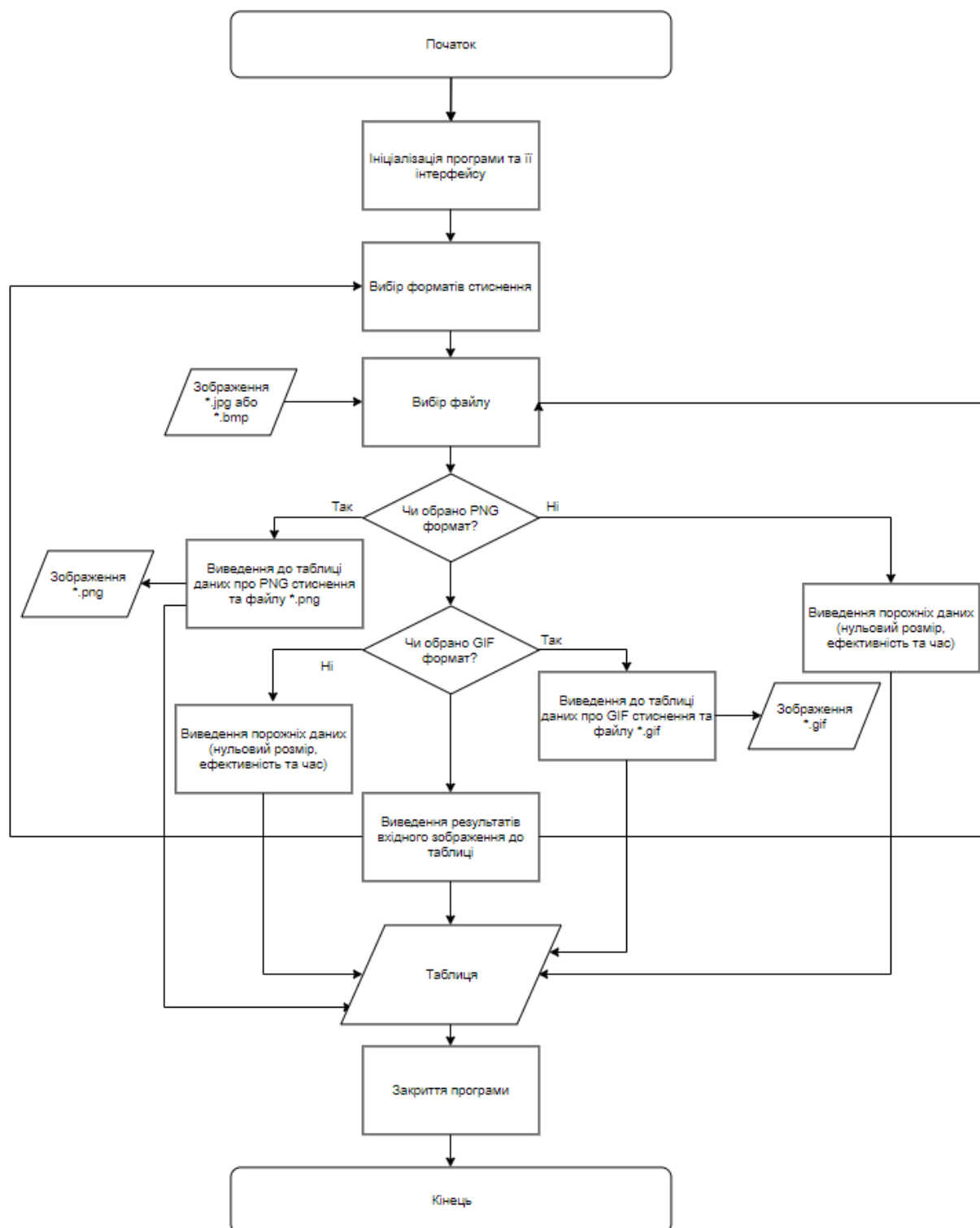
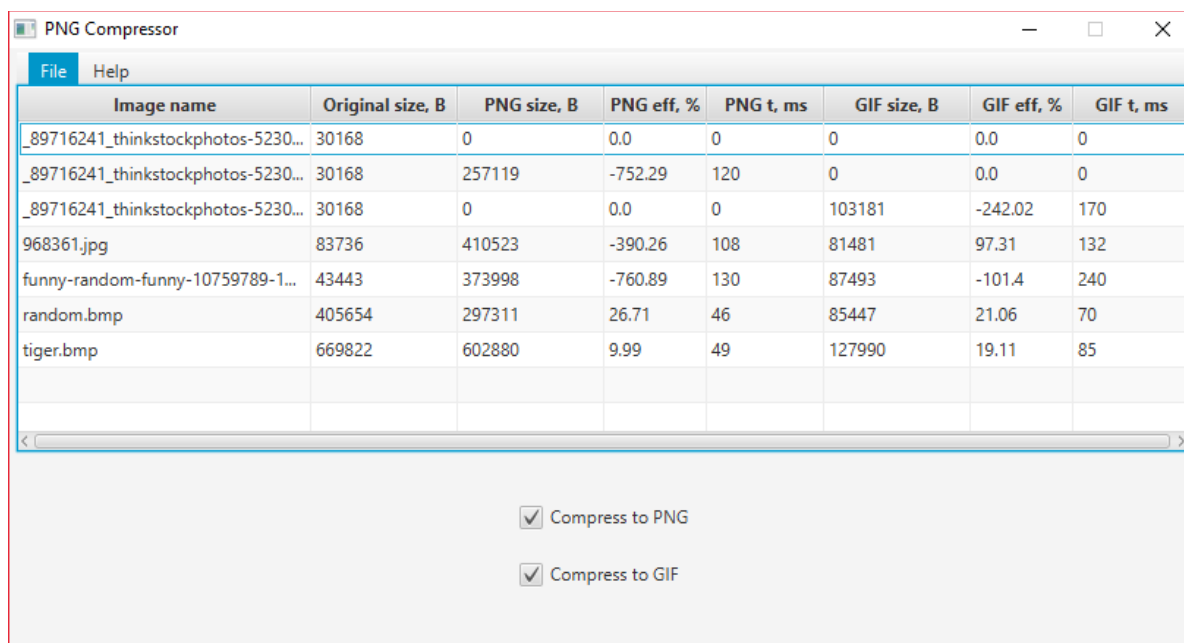


рис. 3.2 – блок-схема алгоритму програми

3.4 Тестування програми

Програма тестується методом вибору різних файлів для компресування та обираючи різні методи. При виборі некоректного файлу (тут, такого, що не є файлом формату .bmp або .jpg) програма закриває поточне діалогове вікно та відкриває нове. Таким чином програма не дозволяє ввести неправильні дані. При виборі коректного зображення програма скомпресує отримане зображення у відповідні файли у тій самій папці, а результати стиснення виводить до таблиці. Таблиця містить у собі дані про оригінальне зображення та отримані, також ефективність стиснення та час стиснення. Якщо стиснений файл має більший розмір, ефективність приймає від'ємне значення. Результати тестування попередньо обраних зображень відображені нижче. Перший файл тестувався тричі (без стиснення, зі стисненням PNG та GIF відповідно). Перші три файли мають формат JPG, останні два – BMP.



The screenshot shows a window titled "PNG Compressor" with a menu bar containing "File" and "Help". Below the menu is a table with 8 columns: "Image name", "Original size, B", "PNG size, B", "PNG eff, %", "PNG t, ms", "GIF size, B", "GIF eff, %", and "GIF t, ms". The table contains 7 rows of data. Below the table, there are two checkboxes: "Compress to PNG" and "Compress to GIF", both of which are checked.

Image name	Original size, B	PNG size, B	PNG eff, %	PNG t, ms	GIF size, B	GIF eff, %	GIF t, ms
_89716241_thinkstockphotos-5230...	30168	0	0.0	0	0	0.0	0
_89716241_thinkstockphotos-5230...	30168	257119	-752.29	120	0	0.0	0
_89716241_thinkstockphotos-5230...	30168	0	0.0	0	103181	-242.02	170
968361.jpg	83736	410523	-390.26	108	81481	97.31	132
funny-random-funny-10759789-1...	43443	373998	-760.89	130	87493	-101.4	240
random.bmp	405654	297311	26.71	46	85447	21.06	70
tiger.bmp	669822	602880	9.99	49	127990	19.11	85

рис. 3.3 – результати тестування та стиснення

ВИСНОВОК:

Метою створення даного продукту є реалізація стиснення файлу у PNG формат та дослідження алгоритму його стиснення та структури самого формату PNG. Цей формат виконує доволі ефективне стиснення, яке за ефективністю хоч і програє тому ж JPEG, але виконує стиснення без втрат. Серед переваг та недоліків, можна виділити наступні:

Переваги

- виконується стиснення без втрат – після декомпресування отримується зображення ідентичне оригінальному.
- має потужний алгоритм стиснення.
- Підтримує truecolor та повноцінний канал прозорості (альфа-канал)
- До 2004 року був вільною альтернативою GIF-стисненню, яке використовує LZW алгоритм, що до 2004 року був недоступний через авторські права (наразі не актуальна перевага, але саме це спричинило розповсюдження PNG).
- Байтові блоки, що відповідають за обраний алгоритм стиснення та фільтрування дозволяють подальше розширення формату через додавання нових алгоритмів.

Недоліки:

- Не підтримує анімацію (існують формати aPNG та MNG, але вони вважаються окремими форматами, хоч і створені на основі PNG).
- Має значно більший розмір стисненого файлу порівняно з форматами стиснення зі втратами

СПИСОК ДЖЕРЕЛ

1. Deutsch P. DEFLATE Compressed Data Format Specification version [Електронний ресурс] / P. Deutsch // Aladdin Enterprises. – 1996. – Режим доступу до ресурсу: <https://www.ietf.org/rfc/rfc1951.txt>.
2. Фелдспар А. An Explanation of the Deflate Algorithm [Електронний ресурс] / Антеус Фелдспар. – 1997. – Режим доступу до ресурсу: <https://zlib.net/feldspar.html>.
3. Portable Network Graphics (PNG) Specification (Second Edition) [Електронний ресурс] / [М. Адлер, Т. Боутел, Д. Боулер та ін.] // W3C. – 2003. – Режим доступу до ресурсу: <https://www.w3.org/TR/PNG/>.

ДОДАТОК

ЛІСТИНГ КОДУ ПРОГРАМИ

Main.java

```
package sample;

package sample;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;
import sample.view.Controller;

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Parent root =
FXMLLoader.load(getClass().getResource("view/sample.fxml"));
        primaryStage.setTitle("PNG Compressor");
        primaryStage.setScene(new Scene(root, 800, 400));
        primaryStage.setResizable(false);
        Controller.getStage(primaryStage);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Controller.java

```
package sample.view;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.scene.control.CheckBox;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Stage;
import sample.model.Compression;
import sample.model.tableData;

import java.io.File;

public class Controller {
    @FXML
    protected CheckBox pngCheck, gifCheck;
    @FXML
```

```

protected TableView<tableData> table;
@FXML
protected TableColumn<tableData, String> imgNameCol;
@FXML
protected TableColumn<tableData, Long> origSizeCol, pngSizeCol,
pngTimeCol, gifSizeCol, gifTimeCol;
@FXML
protected TableColumn<tableData, Double> pngEffCol, gifEffCol;

private static Stage mainStage;

public static void getStage(Stage st) {
    mainStage = st;
}

private ObservableList<tableData> tableValuesList =
FXCollections.observableArrayList();

@FXML
public void handleCompression() {
    Dialogs dialogs = new Dialogs();
    File file = dialogs.FileChooserDialog(mainStage);
    tableValuesList.add(Compression.compression(file,
gifCheck.isSelected(), pngCheck.isSelected()));
    imgNameCol.setCellValueFactory(new
PropertyValueFactory<>("name"));
    origSizeCol.setCellValueFactory(new
PropertyValueFactory<>("origSize"));
    pngSizeCol.setCellValueFactory(new
PropertyValueFactory<>("pngSize"));
    pngEffCol.setCellValueFactory(new
PropertyValueFactory<>("pngEffect"));
    pngTimeCol.setCellValueFactory(new
PropertyValueFactory<>("pngTime"));
    gifSizeCol.setCellValueFactory(new
PropertyValueFactory<>("gifSize"));
    gifEffCol.setCellValueFactory(new
PropertyValueFactory<>("gifEffect"));
    gifTimeCol.setCellValueFactory(new
PropertyValueFactory<>("gifTime"));

    table.setItems(tableValuesList);
}

@FXML
void handleAbout() {
    Dialogs dialogs = new Dialogs();
    dialogs.aboutDialog();
}
}

```

Compression.java

```

package sample.model;

import javax.imageio.ImageIO;

```

```

import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.text.DecimalFormat;

public class Compression {

    public static tableData compression(File origFile, boolean gif,
boolean png) {
        if (origFile == null) return null;
        BufferedImage inputIm;
        File newPngFile, newGifFile;
        tableData element = new tableData();
        element.setName(origFile.getName());
        element.setOrigSize(origFile.length());
        try {
            inputIm = ImageIO.read(origFile);
            DecimalFormat dec = new DecimalFormat("###.##");
            if (gif) {
                long gifTimeStart = System.currentTimeMillis();
                ImageIO.write(inputIm, "gif", newGifFile = new
File(origFile.getPath() + ".gif"));
                element.setGifTime(System.currentTimeMillis() -
gifTimeStart);
                element.setGifSize(newGifFile.length());
                element.setGifEffect(Double.parseDouble(dec.format((100
- ((double) element.getPngSize() / (double) element.getOrigSize()) *
100))));
            }
            if (png) {
                long pngTimeStart = System.currentTimeMillis();
                ImageIO.write(inputIm, "png", newPngFile = new
File(origFile.getPath() + ".png"));
                element.setPngTime(System.currentTimeMillis() -
pngTimeStart);
                element.setPngSize(newPngFile.length());
                element.setPngEffect(Double.parseDouble(dec.format((100
- ((double) element.getPngSize() / (double) element.getOrigSize()) *
100))));
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        return element;
    }
}

```

tableData.java

```

package sample.model;

public class tableData {
    private String name;
    private long origSize;
}

```

```

private long pngSize;
private double pngEffect;
private long pngTime;
private long gifSize;
private double gifEffect;
private long gifTime;

tableData () {
    this.name = "";
    this.origSize = 0;
    this.pngSize = 0;
    this.pngEffect = 0;
    this.pngTime = 0;
    this.gifSize = 0;
    this.gifEffect = 0;
    this.gifTime = 0;
}

public String getName() {
    return name;
}

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

public long getOrigSize() {
    return origSize;
}

public void setOrigSize(long origSize) {
    this.origSize = origSize;
}

public long getPngSize() {
    return pngSize;
}

public void setPngSize(long pngSize) {
    this.pngSize = pngSize;
}

public double getPngEffect() {
    return pngEffect;
}

public void setPngEffect(double pngEffect) {
    this.pngEffect = pngEffect;
}

public long getPngTime() {
    return pngTime;
}

public void setPngTime (long pngTime) {
    this.pngTime = pngTime;
}

```

```

    }

    public long getGifSize() {
        return gifSize;
    }

    public void setGifSize(long gifSize) {
        this.gifSize = gifSize;
    }

    public double getGifEffect() {
        return gifEffect;
    }

    public void setGifEffect(double gifEffect) {
        this.gifEffect = gifEffect;
    }

    public long getGifTime() {
        return this.gifTime;
    }

    public void setGifTime(long gifTime) {
        this.gifTime = gifTime;
    }
}

```

Dialogs.java

```

package sample.view;

import javafx.scene.control.Alert;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

import java.io.File;

class Dialogs {
    File FileChooserDialog(Stage s) {
        try {
            return handleFileChoosing(s);
        } catch (NullPointerException ex) {
            ex.printStackTrace();
        }
        return null;
    }
    private File handleFileChoosing(Stage s) {
        FileChooser choose = new FileChooser();
        choose.setTitle("Choose an image");
        FileChooser.ExtensionFilter filter = new
FileChooser.ExtensionFilter("Images (*.bmp, *.jpg)", "*.bmp", "*.jpg");
        choose.getExtensionFilters().add(filter);
        File file = choose.showOpenDialog(s);
        if (file != null) {

```

```

        if (file.getName().endsWith(".bmp") ||
(file.getName().endsWith(".jpg"))) {
            return file;
        } else {
            file = handleFileChoosing(s);
        }
    }
    return null;
}

void aboutDialog() {
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("About");
    alert.setHeaderText("Made by Sirius Silverclaw - Telegram:
@BlackFolf \n Tested by Despiad\nReviewed by BaLiK");
    alert.showAndWait();
}
}

```