Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра математики факультету інформатики

Фрактальні множини: експерименти з комплексними числами, комп'ютерне моделювання.

Курсова робота за спеціальністю «Прикладна математика»
6.040103

Керівник курсової роботи
к.т.н., доц. Шестюк Н.
_____
(підпис)
“___”_____ 2021 р.
Виконав студент 3-го курсу
Малий Д.К.
_____
(підпис)
“___”_____ 2021 р.

Київ 2021

Календарний план виконання роботи:

Тема: Фрактальні множини: експерименти з комплексними числами, комп'ютерне моделювання.

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	20.10.2020	
2.	Огляд літератури за темою роботи	15.11.2020	
3.	Аналіз літератури за темою роботи.	05.01.2021	
3.	Дослідження результатів отриманих в літературі	30.01.2021	
4.	Аналіз отриманих даних	02.03.2021	
5.	Оформлення курсової роботи	05.05.2021	
6.	Створення презентації	14.05.2021	
7.	Аналіз роботи з керівником	16.05.2021	
8.	Коригування роботи	17.05.2021	
9.	Захист роботи	20.05.2021	

Зміст

ВСТУП

1.1 Актуальність

На сьогоднішній день фрактальні множини набувають все більшої популярності в різних наукових галузях. Вони вже використовуються в економічному моделюванні, моделюванні природних процесів, медицині, інформатиці, радіотехніці та навіть в комп’ютерних іграх. І це не дивно, адже фрактали це геніальний винахід створений самою природою. Безліч природних об'єктів мають фрактальні властивості, починаючи від коралів та броколі і завершуючи сніжинками та блискавкою. Але сам термін «фрактал», було введено відносно недавно, в 1975 році. І я впевнений, що тема фракталів ще недостатньо досліджена і може в майбутньому бути використана в безлічі наукових галузей та досліджень.

Першим, хто почав вивчати фрактальні множини, систематизував знання про них та дав їх визначення був польский математик Бенуа Мандельброт. Саме його книга «How Long Is the Coast of Britain? Statistical Self-Similarity and Fractional Dimension», видана в 1975, і стала відправною точкою у вивченні фрактальних множин. Але, ще задовго до Бенуа Мандельброта, такі відомі вчені як Георг Кантор, Карл Вейєрштрасс, Хельге фон Кох, Поль П'єр Леві та ще багато інших вчених вже давно працювали над самоподібними структурами, але за браком тогочасних технологій вони не змогли відобразити всю красу відкритих ними об'єктів. Саме їх роботи і лягли в основу досліджень Бенуа Мандельброта, результатом яких і стало виділення фракталів як окремої частини геометрії (фрактальна геометрія).

1.2 Мета роботи

Метою даної роботи є систематизація знань про фрактальні множини, побудова фракталів Мандельброта та проведення експериментів, які допоможуть краще зрозуміти природу та сутність данного фракталу.

Розділ 1. Основні означення

1.1 Комплексне число

Комплексним числом називається число вигляду $z = x + iy$, де $x\ y, \in R$. При цьому число x називається дійсною частиною комплексного числа z і позначається Re z, а число y називається уявною частиною комплексного числа z і позначається Im z. Тобто x = Re z, y = Im z. Два комплексні числа $z_1 = x_1 + iy_1$, $z_2 = x_2 + iy_2$ називаються рівними тоді і тільки тоді, коли $x_1 = x_2$, $y_1 = y_2$. Дійсні числа є частинним випадком комплексних, а саме таких, у яких уявна частина дорівнює нулю. Множина комплексних чисел позначається символом C. ([1] ст. 4-5)

1.2 Самоподібний об'єкт

В математиці це об'єкт, що абсолютно або приблизно збігається з частиною себе самого (тобто ціле має ту ж форму, що і одна або більше частин). Багато об'єктів реального світу, наприклад, берегові лінії, мають властивість статистичної самоподібності: їхні частини статистично однорідні в різних шкалах вимірювання. Самоподібність є характеристичною властивістю фракталу ([2] ст. 8).

1.3 Динамічна система

це математична абстракція, призначена для опису і вивчення систем, що еволюціонують з часом. Прикладом можуть служити механічні системи (рухомі групи тіл) або фізичні процеси. [3]

Розділ 2. Фрактали в Комплексній динаміці

2.1 Означення фракталу

Строгого і повного означення фрактала не існує. За означенням, яке наводить Бенуа Мандельброта в своїй книзі, фракталом називається множина, розмірність Хаусдорфа-Безиковича якої є строго більшою за її топологічну розмірність. Тобто будь-яка множина з нецілим значенням D э фракталом([4] ст. 10-11). Але навіть сам Мандельброт не був задоволений даним означенням, адже деякі множини, які розглядаються математиками як фрактали не підпадають під дане визначення. Також в деяких джерелах фрактал описується як нерегулярна, самоподібна структура; фракталами називаються геометричні об'єкти: лінії, поверхні, просторові тіла, що мають сильно порізану форму і володіють властивістю самоподібності.([5] ст. 12).Узагальнено фракталом може називатись фігура, яка володіє деякими з нижче перерахованих властивостей:

- Має нетривіальну структуру незалежно від масштабу, в цьому відмінність від регулярних фігур : якщо ми розглянемо невеликий фрагмент регулярної фігури в дуже великому масштабі, він буде схожий на фрагмент прямої. Для фракталу збільшення масштабу не призводить до спрощення структури, на всіх шкалах ми побачимо однаково складну картину.
- Є самоподібною або наближено самоподібною .
- Має дробову, метричну або фрактальну розмірність, яка перевершує топологічну. ([2] ст. 7)

2.2 Класифікація фракталів

Фрактали прийнято класифікувати на геометричні фрактали, алгебраїчні фрактали та стохастичні фрактали

2.2.1 Геометричні фрактали

Історично перший тип фракталів, який був відомий людству. Цей тип фракталів отримується за допомогою простих геометричних побудов. Для побудови геометричних фракталів зазвичай задають певну “аксіому” (набір геометричних фігур який є основою для побудови фракталу) та “генератор” (певний набір правил який застосовується для перетворення основи, або її частин нескінченну кількість разів).

Прикладами таких фракталів є трикутник Серпінського та сніжинка Коха.

2.2.2 Алгебраїчні фрактали

Друга велика група фракталів. Вона задається за допомогою алгебраїчних форму. Отримують дані фрактали за допомогою нелінійних процесів в n-мірних просторах. Нелінійні, динамічні системи володіють кількома станами. Стан в якому буде динамічна система після певного числа ітерацій залежить від початкових умов. Сталі стани (атрактори) мають певну область початкових станів, з яких система обов'язково перейде в сталий стан. Таким чином простір розбивається на оьласті тяжіння атрактора.

Прикладами таких фракталів є множини Мандельброта та Жюліа.

2.2.3 Стохастичні фрактали

Фрактали при побудові яких у ітеративний процесі випадковим чином змінюються певні параметри. В результаті можна одержати такі об'єкти як гірські ландшафти чи поверхні рельєфу які саме через певну випадковість і будуть здаватися реальними (часто використовується в комп'ютерній графіці)

Прикладом таких фракталів є Броунівський рух ([6] ст 2-3)

2.3 Види самоподібності фракталів

За видами самоподібності фрактали поділяються на точно самоподібні, майже самоподібні та статистично самоподібні.

2.3.1 Точна самоподібність

Найсильніший вид самоподібності, при будь-якому збільшенні фрактал буде мати однаковий зовнішній вигляд. Зазвичай таку самоподібність мають фрактали створені за допомогою ітеративних функцій.

2.3.2 Майже самоподібність (квазі-самоподібність)

При певному збільшенні фрактал виглядає приблизно самоподібним. Збільшена частина містить малі копії цілого фракталу, але в дещо перекручених та вироджених формах. Зазвичай таку самоподібність мають фрактали створені за допомогою рекурентних відношень.

2.3.3 Статистична самоподібність

Найслабший вид самоподібності, такі фрактали містять в собі деякий вид статистичної самоподібності (їх частини є статистично однорідними в різних шкалах виміру, як приклад: берегова лінія). Зазвичай таку самоподібність мають ймовірнісні фрактали.

2.4 Приклади фракталів

2.4.1 Множина Кантора

Фрактальна множина, яка є підмножиною відрізка дійсних чисел [0,1]. Будується даний фрактал за допомогою вилучення серединних сегментів з інтервалу [0,1]. Довжина вилученого сегменту дорівнює ⅓ початкової довжин відрізку і видаляється вона в центрі відрізку. Після цього з отриманими двома відрізками виконуються ті ж самі дії і так нескінченну кількість разів. Множина Кантора складається з тих точок, що залишилися після всіх видалень. ([7] ст. 222-223)

Рисунок 2.4.1 - Множина Кантора

2.4.1 Трикутник Серпінського

Це фрактал, який э одним з двовимірних аналогів множини Кантора. Даний фрактал отримується таким чином:

1. За основу береться суцільний (чорний) трикутник
2. З нього видаляється центральний трикутник (трикутник отриманий сполученням середніх ліній)
3. Далі даний процес повторюється нескінченну кількість разів з усіми чорними трикунтиками, які утворилися після минулого кроку

Вищеописаний метод отримання трикутника Серпінського (рис. 2.4.1) є геометричним.

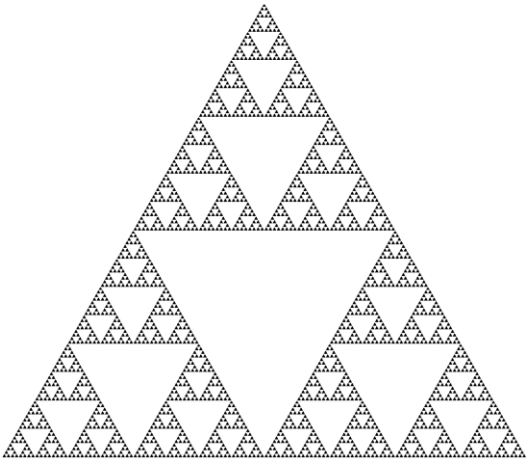

Рисунок 2.4.2 – трикутник Серпінського

Також даний фрактал можна побудувати за допомогою стохастичної процедури (звичайно в даному випадку фрактал вже не буде точно самоподібним, як при геометричній побудові). Для такої побудови послідовність дій така:

1. На площині накреслимо рівносторонній трикутник. Кожній з вершин поставимо у відповідність два числа з грального кубика (наприклад першій вершині відповідають числа 1, 2, другій - 3, 4, третій 5,6)
2. Оберемо довільну точку *a* всередині трикутника
3. Підкинемо кубик. Центр відрізку від *a* до вершини, відповідний номер якої випав на кубику, стане новою точкою *a.* Далі будемо повторювати крок 3 досить велику кількість разів.

Множина таких точок *a* після великої кількості ітерацій і утворить трикутрник Серпінського (рис. 2.4.3)

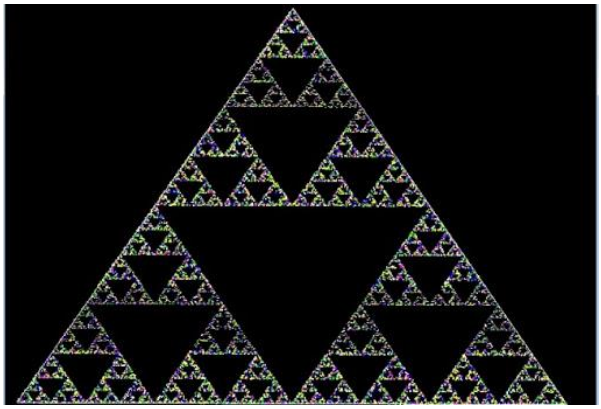

Рисунок 2.4.3 - трикутник Серпінського побудований стохастично

Розмірність Хаусдорфа трикутника Серпінського приблизно дорівнює 1,585. ([8] ст 274-276)

2.4.3 Крива Коха

Фрактальна крива, описана Хельге фон Кохом. Особливістю даної кривої є те, що вона не має дотичних, тобто є ніде не диференційованою, але при цьому є всюди неперервною.

Основою даної кривої є одиничний відрізок(рис 2.4.4). Розділимо його на три частини. Замінимо центральний сегмент на рівносторонній трикутник без цього сегменту (рис 2.4.5)

Рисунок 2.4.4 - основа

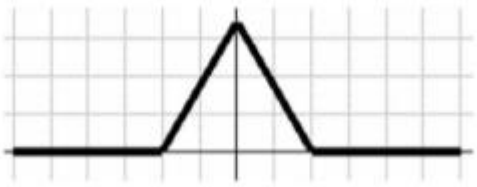

Рисунок 2.4.5 - генератор

Далі для кожної ланки отриманої ламаної зробимо те саме(рис. 2.4.6) і будемо повторювати даний процес нескінченність разів.([2] ст. 5-6)

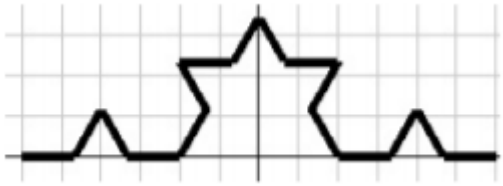

Рисунок 2.4.6 - друга ітерація

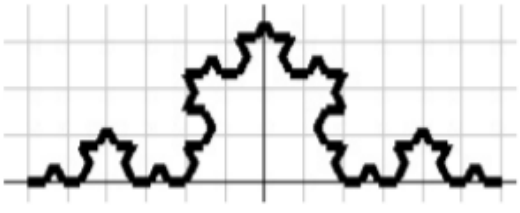

Рисунок 2.4.7 - крива Коха

А якщо за основу взяти рівносторонній трикутник, то замість кривої Коха, отримаємо фрактал сніжинка Коха.

2.5 Використання фракталів

2.5.1 В радіотехніці

Саме завдяки фрактальній формі вдалося значно зменшити розміри антен у мобільних. В 1999 було налагоджено перший випуск фрактальних антен і зараз вони використовуються майже в кожному телефоні. ([9] ст. 78—83.)

2.5.2 В інформатиці

Фрактали використовуються в інформатиці для стиснення великих растрових зображень. На даний момент цей алгоритм не набув широкого розповсюдження і допрацьовується, через те, що процес стиснення є досить складним і довгим для обчислення.[10]

2.5.3 В економіці

Фрактали широко використовуються для економічного моделювання поведінки фінансових ринків.

Розділ 3. Множина Жюліа

3.1. Загальні відомості

Це динамічна система, задана на комплексній площині. Множина Жюліа певної функції f є границею множини точок z, які прямують до нескінченності при ітеруванні $f(z) = z^2 + C$, де C - певне комплексне число, а z_0 комплексне число яке відповідає координатам відповідної точки на площині

Побудова множин Жюліа здійснюється таким чином :

1. Обирається комплексне число C, та кількість ітерацій m (в ідеальному випадку кількість ітерацій має дорівнювати нескінченності)

2. Для кожної точки простору обраховується значення $f(z_m)$ після заданої кількості ітерацій. Якщо значення точки впродовж всіх ітерацій не прямує до нескінченності, а залишається в певній, обмеженій області, то дана точка належить множині Жуліа. (якщо $|f(z_a)| > 2$, де a - номер ітерації від 0 до m, то данне число точно буде прямувати в до нескінченності і не належить множині Жюліа). ([5] ст. 66-69)

Нижче на рисунках зображені приклади множин Жюліа з різними параметрами (рис 2.4.8, 2.4.9). Вони виглядають красиво через додавання кольору. На даних малюнках колір точки залежить від кількості ітерацій, які треба було зробити щоб визначити, що точка не належить множині Жуліа. На кольоровій схемі (рис 2.4.10) чим правіше стоїть колір, тим більшу кількість ітерацій потрібно було зробити, щоб визначити чи належить множині Жюліа відповідна точка. Чорні області це точки, які належать множині Жуліа.

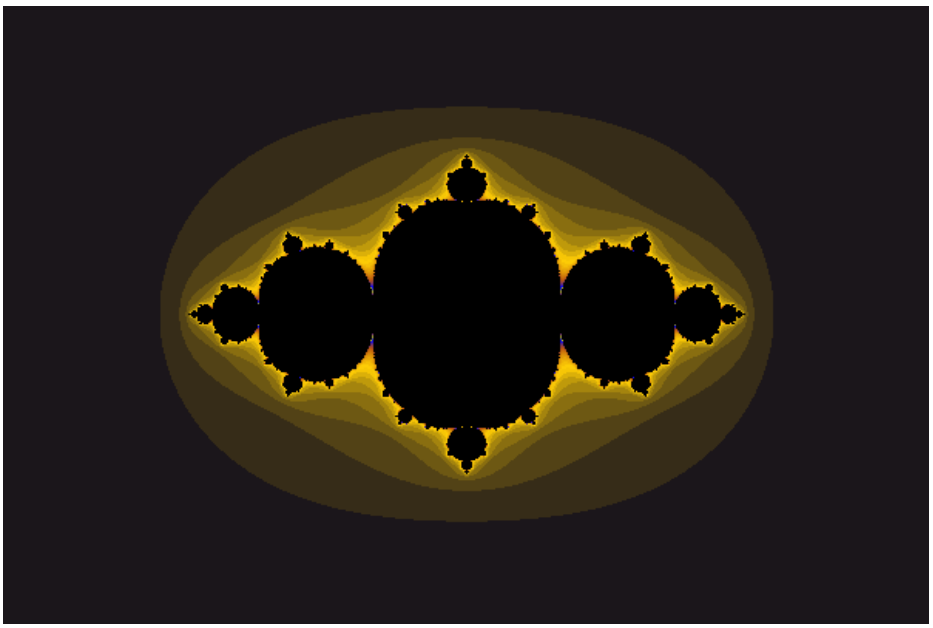

Рисунок 3.1.1 - множина Жуліа з *C= -0,75 , m = 120*

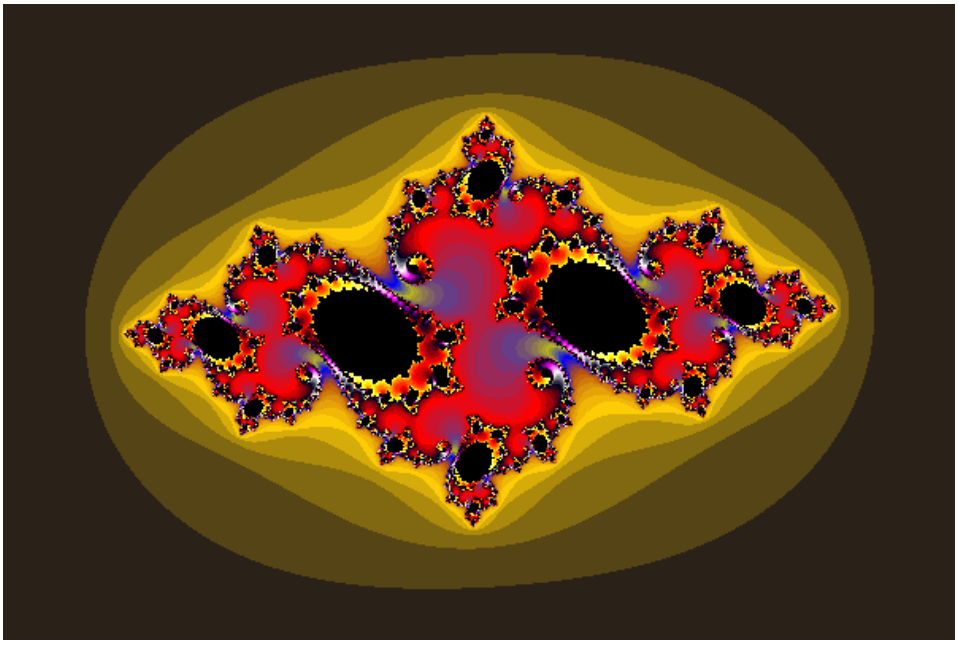

Рисунок 3.1.2 - множина Жуліа з *C= -0,75-0,1i , m = 77*

Рисунок 3.1.3 - кольорова схема

3.2. Експерименти зі зміною ступеня z

Нижче на рисунку показано перетворення множини Жуліа (рис 3.1.3). Як бачимо при збільшенні ступеня z, множина поступово набуває форму кола.

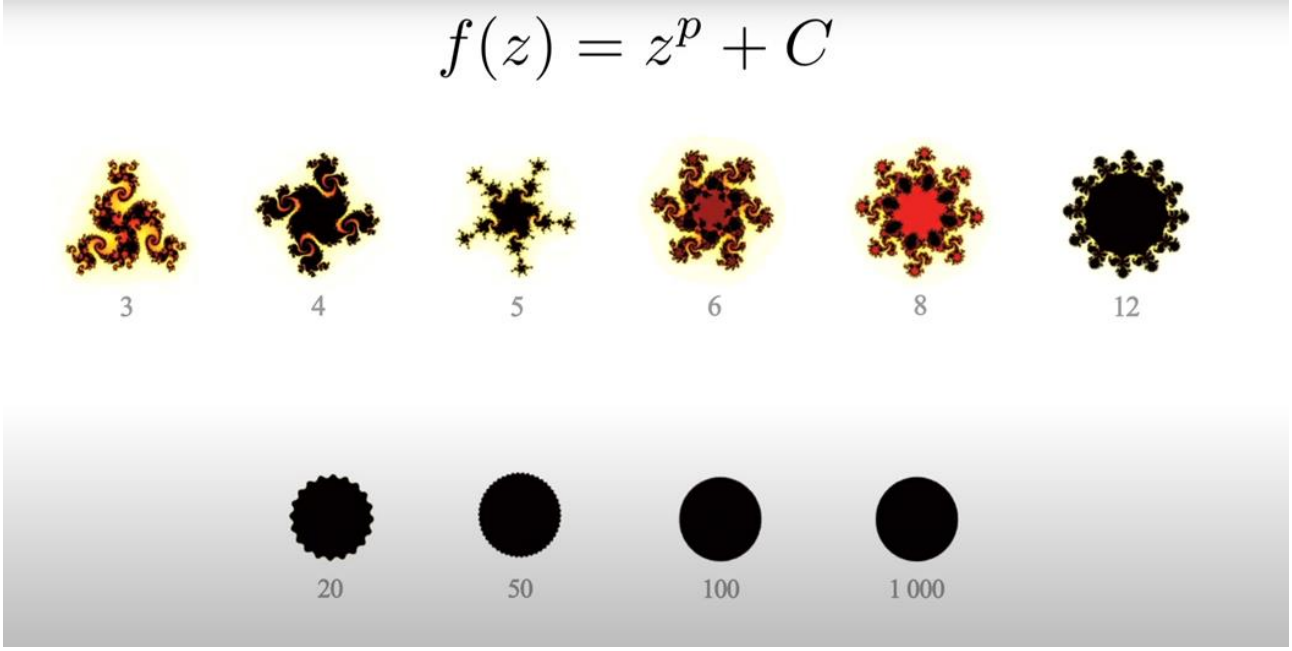

Рисунок 3.1.3 - експеримент зі зміною ступеня

Розділ 4. Множина Мандельброта

4.1. Загальні відомості

Це динамічна система, задана на комплексній площині. Множина Мандельброта(рис 4.1.1) складається з точок *c* для яких $f_c(z) = z^2 + c$ не розходиться при ітеруванні. Тобто, якщо значення точки впродовж всіх ітерацій не прямує до нескінченності, а залишається в певній, обмеженій області. Зображення множини Мандельброта створюється так само за допомогою кольорів, як і множини Жуліа.

Якщо *c* в формулі залишити сталим, а початкове значення *z* зробити змінним, то отримаємо відповідну множину Жуліа для точки *c*

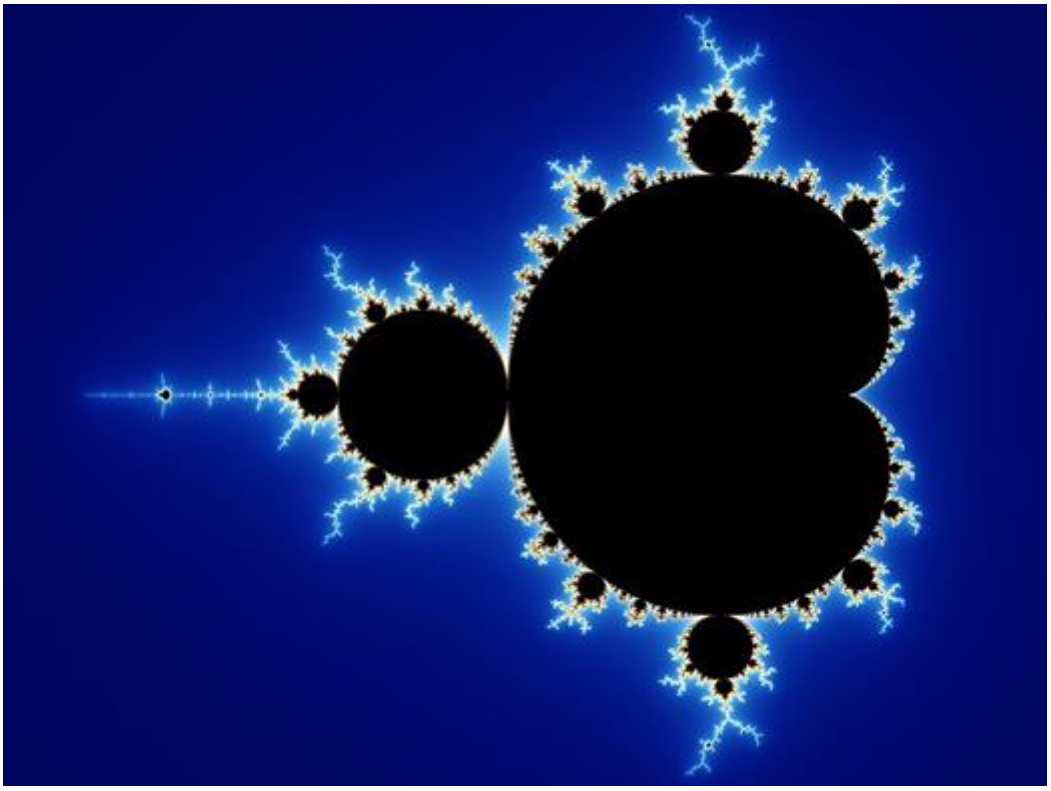

Рисунок 4.1.1 - Множина мандельброта

4.2 Експерименти з множинами Мандельброта

Подальші експерименти проводилися за допомогою програми “Mandelbrot Experiments”. Програмний код буде наведено в додатку А.

4.2.1 Експерименти зі зміною ступеня z

Нижче на рисунках показано перетворення множини Мандельброта (рис 4.2.1 - 4.2.5). Як бачимо при збільшенні ступеня z, Множина Мандельброта поводить себе так само як і множина Жюліа , і поступово набуває форму кола.

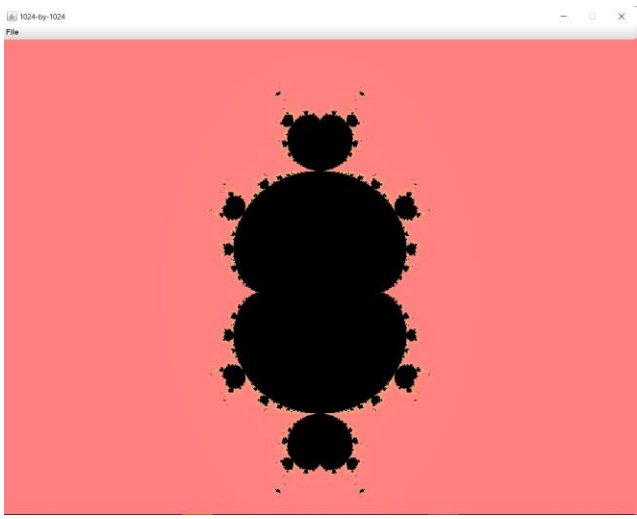

Рисунок 4.2.1 - ступінь 2

Рисунок 4.2.2 - ступінь 3

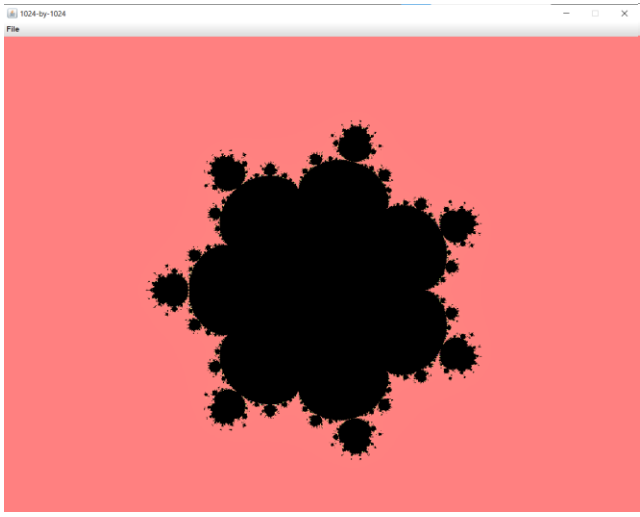

Рисунок 4.2.3 - ступінь 7

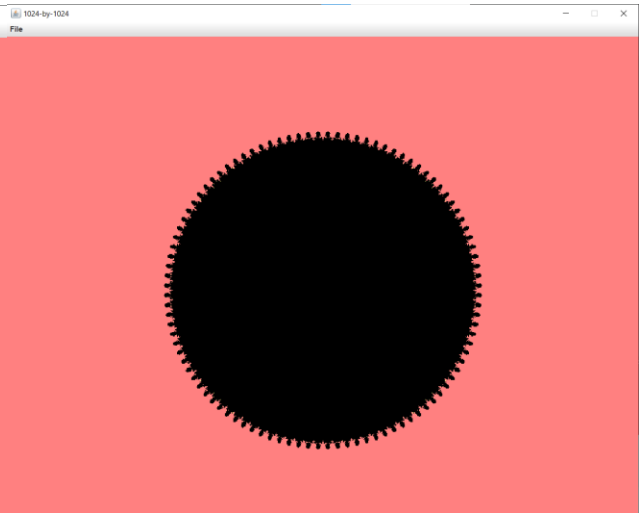

Рисунок 4.2.4 - ступінь 100

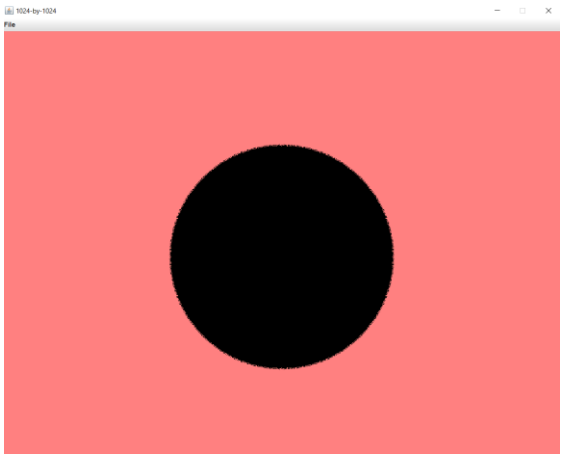

Рисунок 4.2.5 - ступінь 500

4.2.2 Експерименти зі зміною функції

Тут показано що буде відбуватися з множиною, якщо до z^2 застосовувати різні тригонометричні функції (рис 4.2.6 - 4.2.8)

$$f(z) = \sin(z^2) + C$$

Рисунок 4.2.6

$$f(z) = tg(z^2) + C$$

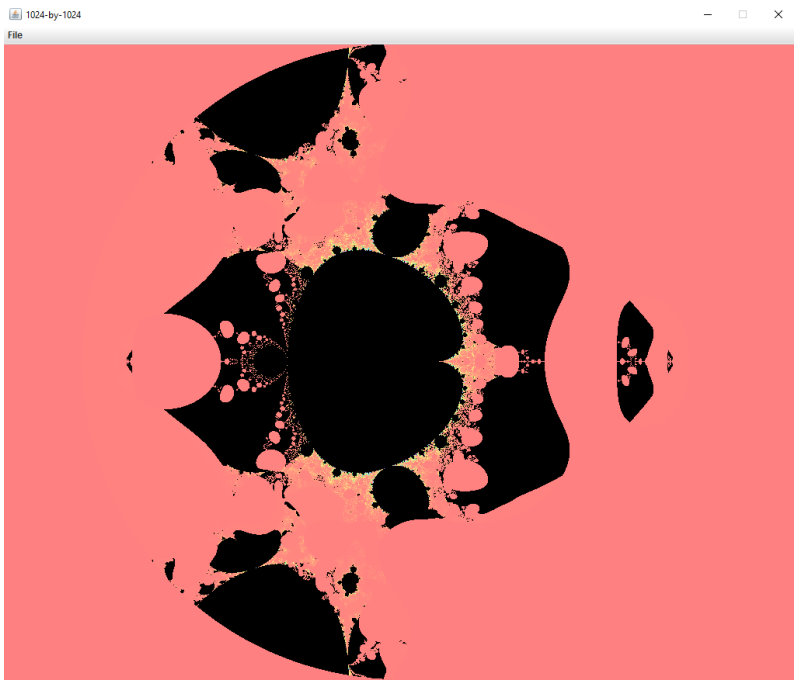

Рисунок 4.2.7

$f(z) = \cos(z^2) + C$

Рисунок 4.2.8

ВИСНОВКИ

Під час роботи, мною було досліджено фрактальні множини. Було розглянуто класифікацію фракталів та їх актуальність і шляхи практичного використання. Найбільшу увагу було приділено множинам Мандельброта та Жуліа. Я набув практичний досвід дослідження фрактальних множин (множини Мандельброта) за допомогою програмних засобів.

СПИСОК ЛІТЕРАТУРИ

1. С. А. Щоголєв, “Комплексні числа" / Щоголєв Сергій Авенірович. — Одеський національний університет імені І. І. Мечникова, 2015р. — 44 ст. — (електронна версія : http://dspace.onu.edu.ua:8080/bitstream/123456789/22779/1/red_compl4.pdf)
2. І. Ю. Адашевська, О. О. Краєвська “Самоподібність як характеристична властивість фракталу. Фрактальна (дробова) розмірність Хаусдорфа” / Адашевська Ірина Юріївна, Краєвська Олена Олександрівна — Національний технічний університет «ХПІ» м. Харків, 2019 р. — 10 ст. — (електронна версія : http://repository.kpi.kharkov.ua/handle/KhPI-Press/44138)
3. “Динамічна система” / Вікіпедія (версія від 14 грудня 2020 року, дата цитування 17.05.2021) — (постійне посилання: https://uk.wikipedia.org/w/index.php?title=%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D1%96%D1%87%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0&oldid=30309806 , ID версії сторінки: 30309806)
4. Б.К. Мандельброт, “Фрактальная геометрия природы” / Мандельброт Бенуа Карлович — Московський інститут комп'ютерних досліджень 2002 р. — 656 ст.
5. С.В. Божокін, Д.А. Паршин, “Фракталы и мультифракталы” / Божокін Сергій Валентинович, Паршин Олексій Миколайович — Іжевський науково-дослідницький центр, 2001р. — 128 ст.
6. Т.Ж. Надригайло , О.С. Тітаренко, “Програмні засоби генерування фрактальних множин: Математичне моделювання 2(21) - 2009” — Дніпровський державний технічний університет, 2009 р. — 5 ст. — (електронна версія : http://178.219.93.18:8080/Portal/Data/74/66/7st-0-2.pdf)

7. S.A. Morris, “Topology without tears” / Sidney Allen Morris — версія від 28 червня, 2020 — 704 ст. — (електронна версія: topbook.pdf (topologywithouttears.net))
8. А.М. Ширяев, “Основы стохастической финансовой математики”, том 1 / Альберт Миколайович Ширяев — Москва: ФАЗИС, 1998 р. — 512 ст.
9. В.І.Слюсар, “Фрактальные антенны. Принципиально новый тип «ломаных» антенн.”/ Журнал “Электроника: наука, технология, бизнес”. № 5. — 2007р.
10. Р.А. Зубко “Стиснення зображень фрактальним методом” / Зубко Роман Анатолійович — 2014 р. — 6 ст — (електронна версія : http://www.irbis-nbuv.gov.ua/cgi-bin/irbis_nbuv/cgiirbis_64.exe?C21COM=2&I21DBN=UJRN&P21DBN=UJRN&IMAGE_FILE_DOWNLOAD=1&Image_file_name=PDF/Vejpte_2014_6%282%29__5.pdf)

Додаток А

Програмний код “Mandelbrot Experiments”

```
import java.awt.*;

public class Main {

   public static void main(String[] args) {
       double xc = 0.0; // center point for x (def
0.0)
       double yc = 0.0; // center point for y (def
0.0)
       double size = 5.0; //scale (def 4.0)

       int n = 1024;    // create n-by-n image
       int max = 120;    // maximum number of
iterations

       Picture picture = new Picture(n, n);

       for (int i = 0; i < n; i++) {
           for (int j = 0; j < n; j++) {
               double x0 = xc - size / 2 + size * i /
n;
               double y0 = yc - size / 2 + size * j /
n;
               Complex z0 = new Complex(x0, y0);

               if (max - mandelbrot(z0, max) == 0) {
                   Color color = new
Color(0x00000000);
                   picture.set(i, n - 1 - j, color);
               } else {
                   Color color = new
Color(Color.HSBtoRGB((float) mandelbrot(z0, max) /
max, 0.5f, 1));
                   picture.set(i, n - 1 - j, color);
               }
           }
       }
       picture.show();
   }

   // return number of iterations to check if c = a +
ib is in Mandelbrot set
```

```
    public static int mandelbrot(Complex z0, int max)
{
        Complex z = z0;
        for (int t = 0; t < max; t++) {
            if (z.abs() > 2.0) {
                return t;
            }
            z = pow(z, z0, 2); // experiment with
formula

        }
        return max;
    }


    // pow of Complex nums
    public static Complex pow(Complex z, Complex z0,
int num) {
        Complex answer = z;
        for (int i = 0; i < num; i++) {
            answer = answer.times(z);
        }
        return answer.plus(z0);
    }
}
```

```
import java.util.Objects;
/**
*  The {@code Complex} class represents a complex
number.
*  Complex numbers are immutable: their values cannot
be changed after they
*  are created.
*  It includes methods for addition, subtraction,
multiplication, division,
*  conjugation, and other common functions on complex
numbers.
*  <p>
*  This computes correct results if all arithmetic
performed is
*  without floating-point rounding error or
arithmetic overflow.
*  In practice, there will be floating-point rounding
error.
```

```
*  <p>
*  For additional documentation, see <a
href="https://algs4.cs.princeton.edu/99scientific">Se
ction 9.9</a> of
*  <i>Algorithms, 4th Edition</i> by Robert Sedgewick
and Kevin Wayne.
*
*  @author Robert Sedgewick
*  @author Kevin Wayne
*/
public class Complex {
   private final double re;   // the real part
   private final double im;   // the imaginary part

   // create a new object with the given real and
imaginary parts
   public Complex(double real, double imag) {
       re = real;
       im = imag;
   }

   // return a string representation of the invoking
Complex object
   public String toString() {
       if (im == 0) return re + "";
       if (re == 0) return im + "i";
       if (im <  0) return re + " - " + (-im) + "i";
       return re + " + " + im + "i";
   }

   // return abs/modulus/magnitude
   public double abs() {
       return Math.hypot(re, im);
   }

   // return abs/modulus/magnitude
   public static double abs(Complex complex) {
       return Math.hypot(complex.re, complex.im);
   }

   // return angle/phase/argument, normalized to be
between -pi and pi
   public double phase() {
       return Math.atan2(im, re);
   }
```

```
   // return a new Complex object whose value is
(this + b)
   public Complex plus(Complex b) {
       Complex a = this;             // invoking
object
       double real = a.re + b.re;
       double imag = a.im + b.im;
       return new Complex(real, imag);
   }

   // return a new Complex object whose value is
(this - b)
   public Complex minus(Complex b) {
       Complex a = this;
       double real = a.re - b.re;
       double imag = a.im - b.im;
       return new Complex(real, imag);
   }

   // return a new Complex object whose value is
(this * b)
   public Complex times(Complex b) {
       Complex a = this;
       double real = a.re * b.re - a.im * b.im;
       double imag = a.re * b.im + a.im * b.re;
       return new Complex(real, imag);
   }

   // return a new object whose value is (this *
alpha)
   public Complex scale(double alpha) {
       return new Complex(alpha * re, alpha * im);
   }

   // return a new Complex object whose value is the
conjugate of this
   public Complex conjugate() {
       return new Complex(re, -im);
   }

   // return a new Complex object whose value is the
reciprocal of this
   public Complex reciprocal() {
       double scale = re*re + im*im;
       return new Complex(re / scale, -im / scale);
   }
```

```
    // return the real or imaginary part
    public double re() { return re; }
    public double im() { return im; }

    // return a / b
    public Complex divides(Complex b) {
        Complex a = this;
        return a.times(b.reciprocal());
    }

    // return a new Complex object whose value is the
complex exponential of this
    public Complex exp() {
        return new Complex(Math.exp(re) *
Math.cos(im), Math.exp(re) * Math.sin(im));
    }

    // return a new Complex object whose value is the
complex sine of this
    public Complex sin() {
        return new Complex(Math.sin(re) *
Math.cosh(im), Math.cos(re) * Math.sinh(im));
    }

    // return a new Complex object whose value is the
complex cosine of this
    public Complex cos() {
        return new Complex(Math.cos(re) *
Math.cosh(im), -Math.sin(re) * Math.sinh(im));
    }

    // return a new Complex object whose value is the
complex tangent of this
    public Complex tan() {
        return sin().divides(cos());
    }



    // a static version of plus
    public static Complex plus(Complex a, Complex b) {
        double real = a.re + b.re;
        double imag = a.im + b.im;
        Complex sum = new Complex(real, imag);
        return sum;
```

```
    }

    // See Section 3.3.
    public boolean equals(Object x) {
        if (x == null) return false;
        if (this.getClass() != x.getClass()) return
false;
        Complex that = (Complex) x;
        return (this.re == that.re) && (this.im ==
that.im);
    }

    // See Section 3.3.
    public int hashCode() {
        return Objects.hash(re, im);
    }

    // sample client for testing
    public static void main(String[] args) {
        Complex a = new Complex(5.0, 6.0);
        Complex b = new Complex(-3.0, 4.0);

        System.out.println("a            = " + a);
        System.out.println("b            = " + b);
        System.out.println("Re(a)        = " +
a.re());
        System.out.println("Im(a)        = " +
a.im());
        System.out.println("b + a        = " +
b.plus(a));
        System.out.println("a - b        = " +
a.minus(b));
        System.out.println("a * b        = " +
a.times(b));
        System.out.println("b * a        = " +
b.times(a));
        System.out.println("a / b        = " +
a.divides(b));
        System.out.println("(a / b) * b  = " +
a.divides(b).times(b));
        System.out.println("conj(a)      = " +
a.conjugate());
        System.out.println("|a|          = " +
a.abs());
        System.out.println("tan(a)       = " +
a.tan());
```

```
    }

}


import java.awt.Color;
import java.awt.FileDialog;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.net.URL;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.KeyStroke;


/**
* This class provides methods for manipulating
individual pixels of
* an image using the RGB color format. The alpha
component (for transparency)
* is not currently supported.
* The original image can be read from a {@code PNG},
{@code GIF},
* or {@code JPEG} file or the user can create a blank
image of a given dimension.
* This class includes methods for displaying the
image in a window on
* the screen or saving it to a file.
* <p>
* Pixel (<em>col</em>, <em>row</em>) is column
<em>col</em> and row <em>row</em>.
* By default, the origin (0, 0) is the pixel in the
top-left corner,
* which is a common convention in image processing.
```

```
* The method {@link #setOriginLowerLeft()} change the
origin to the lower left.
* <p>
* The {@code get()} and {@code set()} methods use
{@link Color} objects to get
* or set the color of the specified pixel.
* The {@code getRGB()} and {@code setRGB()} methods
use a 32-bit {@code int}
* to encode the color, thereby avoiding the need to
create temporary
* {@code Color} objects. The red (R), green (G), and
blue (B) components
* are encoded using the least significant 24 bits.
* Given a 32-bit {@code int} encoding the color, the
following code extracts
* the RGB components:
* <blockquote><pre>
*  int r = (rgb &gt;&gt; 16) &amp; 0xFF;
*  int g = (rgb &gt;&gt;  8) &amp; 0xFF;
*  int b = (rgb &gt;&gt;  0) &amp; 0xFF;
*  </pre></blockquote>
* Given the RGB components (8-bits each) of a color,
* the following statement packs it into a 32-bit
{@code int}:
* <blockquote><pre>
*  int rgb = (r &lt;&lt; 16) + (g &lt;&lt; 8) + (b
&lt;&lt; 0);
* </pre></blockquote>
* <p>
* A <em>W</em>-by-<em>H</em> picture uses ~ 4 <em>W
H</em> bytes of memory,
* since the color of each pixel is encoded as a 32-
bit <code>int</code>.
* <p>
* For additional documentation, see
* <a
href="https://introcs.cs.princeton.edu/31datatype">Se
ction 3.1</a> of
* <i>Computer Science: An Interdisciplinary
Approach</i>
* by Robert Sedgewick and Kevin Wayne.
* See {@link } for a version that supports grayscale
images.
*
* @author Robert Sedgewick
* @author Kevin Wayne
```

```
*/
public final class Picture implements ActionListener
{
   private BufferedImage image;               // the
rasterized image
   private JFrame frame;                      // on-
screen view
   private String filename;                   // name
of file
   private boolean isOriginUpperLeft = true;  //
location of origin
   private final int width, height;           //
width and height

   /**
    * Creates a {@code width}-by-{@code height}
picture, with {@code width} columns
    * and {@code height} rows, where each pixel is
black.
    *
    * @param width  the width of the picture
    * @param height the height of the picture
    * @throws IllegalArgumentException if {@code
width} is negative or zero
    * @throws IllegalArgumentException if {@code
height} is negative or zero
    */
   public Picture(int width, int height) {
       if (width <= 0) throw new
IllegalArgumentException("width must be positive");
       if (height <= 0) throw new
IllegalArgumentException("height must be positive");
       this.width = width;
       this.height = height;
       image = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
       // set to TYPE_INT_ARGB here and in next
constructor to support transparency
   }

   /**
    * Creates a new picture that is a deep copy of
the argument picture.
    *
    * @param picture the picture to copy
```

```
     * @throws IllegalArgumentException if {@code
picture} is {@code null}
     */
    public Picture(Picture picture) {
        if (picture == null) throw new
IllegalArgumentException("constructor argument is
null");

        width = picture.width();
        height = picture.height();
        image = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
        filename = picture.filename;
        isOriginUpperLeft = picture.isOriginUpperLeft;
        for (int col = 0; col < width(); col++)
            for (int row = 0; row < height(); row++)
                image.setRGB(col, row,
picture.image.getRGB(col, row));
    }

    /**
     * Creates a picture by reading an image from a
file or URL.
     *
     * @param name the name of the file (.png, .gif,
or .jpg) or URL.
     * @throws IllegalArgumentException if cannot read
image
     * @throws IllegalArgumentException if {@code
name} is {@code null}
     */
    public Picture(String name) {
        if (name == null) throw new
IllegalArgumentException("constructor argument is
null");
        if (name.length() == 0) throw new
IllegalArgumentException("constructor argument is the
empty string");

        this.filename = name;
        try {
            // try to read from file in working
directory
            File file = new File(name);
            if (file.isFile()) {
                image = ImageIO.read(file);
```

```
            } else {

                // resource relative to .class file
                URL url =
getClass().getResource(filename);

                // resource relative to classloader
root
                if (url == null) {
                    url =
getClass().getClassLoader().getResource(name);
                }

                // or URL from web
                if (url == null) {
                    url = new URL(name);
                }

                image = ImageIO.read(url);
            }

            if (image == null) {
                throw new
IllegalArgumentException("could not read image: " +
name);
            }

            width = image.getWidth(null);
            height = image.getHeight(null);
        } catch (IOException ioe) {
            throw new IllegalArgumentException("could
not open image: " + name, ioe);
        }
    }

    /**
     * Creates a picture by reading the image from a
PNG, GIF, or JPEG file.
     *
     * @param file the file
     * @throws IllegalArgumentException if cannot read
image
     * @throws IllegalArgumentException if {@code
file} is {@code null}
     */
    public Picture(File file) {
```

```
        if (file == null) throw new
IllegalArgumentException("constructor argument is
null");

        try {
            image = ImageIO.read(file);
        } catch (IOException ioe) {
            throw new IllegalArgumentException("could
not open file: " + file, ioe);
        }
        if (image == null) {
            throw new IllegalArgumentException("could
not read file: " + file);
        }
        width = image.getWidth(null);
        height = image.getHeight(null);
        filename = file.getName();
    }

    /**
     * Returns a {@link JLabel} containing this
picture, for embedding in a {@link JPanel},
     * {@link JFrame} or other GUI widget.
     *
     * @return the {@code JLabel}
     */
    public JLabel getJLabel() {
        if (image == null) return null;         // no
image available
        ImageIcon icon = new ImageIcon(image);
        return new JLabel(icon);
    }

    /**
     * Sets the origin to be the upper left pixel.
This is the default.
     */
    public void setOriginUpperLeft() {
        isOriginUpperLeft = true;
    }

    /**
     * Sets the origin to be the lower left pixel.
     */
    public void setOriginLowerLeft() {
        isOriginUpperLeft = false;
```

```
    }

    /**
     * Displays the picture in a window on the screen.
     */

    // getMenuShortcutKeyMask() deprecated in Java 10
but its replacement
    // getMenuShortcutKeyMaskEx() is not available in
Java 8
    @SuppressWarnings("deprecation")
    public void show() {

        // create the GUI for viewing the image if
needed
        if (frame == null) {
            frame = new JFrame();

            JMenuBar menuBar = new JMenuBar();
            JMenu menu = new JMenu("File");
            menuBar.add(menu);
            JMenuItem menuItem1 = new JMenuItem("
Save...   ");
            menuItem1.addActionListener(this);

menuItem1.setAccelerator(KeyStroke.getKeyStroke(KeyEv
ent.VK_S,

Toolkit.getDefaultToolkit().getMenuShortcutKeyMask())
);
            menu.add(menuItem1);
            frame.setJMenuBar(menuBar);


            frame.setContentPane(getJLabel());
            //
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOS
E);
            if (filename == null) frame.setTitle(width
+ "-by-" + height);
            else frame.setTitle(filename);
            frame.setResizable(false);
            frame.pack();
            frame.setVisible(true);
```

```
        }

        // draw
        frame.repaint();
    }

    /**
     * Returns the height of the picture.
     *
     * @return the height of the picture (in pixels)
     */
    public int height() {
        return height;
    }

    /**
     * Returns the width of the picture.
     *
     * @return the width of the picture (in pixels)
     */
    public int width() {
        return width;
    }

    private void validateRowIndex(int row) {
        if (row < 0 || row >= height())
            throw new IllegalArgumentException("row
index must be between 0 and " + (height() - 1) + ": "
+ row);
    }

    private void validateColumnIndex(int col) {
        if (col < 0 || col >= width())
            throw new IllegalArgumentException("column
index must be between 0 and " + (width() - 1) + ": "
+ col);
    }

    /**
     * Returns the color of pixel ({@code col}, {@code
row}) as a {@link java.awt.Color}.
     *
     * @param col the column index
     * @param row the row index
     * @return the color of pixel ({@code col}, {@code
row})
```

```
     * @throws IllegalArgumentException unless both
{@code 0 <= col < width} and {@code 0 <= row <
height}
     */
    public Color get(int col, int row) {
        validateColumnIndex(col);
        validateRowIndex(row);
        int rgb = getRGB(col, row);
        return new Color(rgb);
    }

    /**
     * Returns the color of pixel ({@code col}, {@code
row}) as an {@code int}.
     * Using this method can be more efficient than
{@link #get(int, int)} because
     * it does not create a {@code Color} object.
     *
     * @param col the column index
     * @param row the row index
     * @return the integer representation of the color
of pixel ({@code col}, {@code row})
     * @throws IllegalArgumentException unless both
{@code 0 <= col < width} and {@code 0 <= row <
height}
     */
    public int getRGB(int col, int row) {
        validateColumnIndex(col);
        validateRowIndex(row);
        if (isOriginUpperLeft) return
image.getRGB(col, row);
        else return image.getRGB(col, height - row -
1);
    }

    /**
     * Sets the color of pixel ({@code col}, {@code
row}) to given color.
     *
     * @param col   the column index
     * @param row   the row index
     * @param color the color
     * @throws IllegalArgumentException unless both
{@code 0 <= col < width} and {@code 0 <= row <
height}
```

```
     * @throws IllegalArgumentException if {@code
color} is {@code null}
     */
    public void set(int col, int row, Color color) {
        validateColumnIndex(col);
        validateRowIndex(row);
        if (color == null) throw new
IllegalArgumentException("color argument is null");
        int rgb = color.getRGB();
        setRGB(col, row, rgb);
    }

    /**
     * Sets the color of pixel ({@code col}, {@code
row}) to given color.
     *
     * @param col the column index
     * @param row the row index
     * @param rgb the integer representation of the
color
     * @throws IllegalArgumentException unless both
{@code 0 <= col < width} and {@code 0 <= row <
height}
     */
    public void setRGB(int col, int row, int rgb) {
        validateColumnIndex(col);
        validateRowIndex(row);
        if (isOriginUpperLeft) image.setRGB(col, row,
rgb);
        else image.setRGB(col, height - row - 1, rgb);
    }

    /**
     * Returns true if this picture is equal to the
argument picture.
     *
     * @param other the other picture
     * @return {@code true} if this picture is the
same dimension as {@code other}
     * and if all pixels have the same color; {@code
false} otherwise
     */
    public boolean equals(Object other) {
        if (other == this) return true;
        if (other == null) return false;
```

```
        if (other.getClass() != this.getClass())
return false;
        Picture that = (Picture) other;
        if (this.width() != that.width()) return
false;
        if (this.height() != that.height()) return
false;
        for (int col = 0; col < width(); col++)
            for (int row = 0; row < height(); row++)
                if (this.getRGB(col, row) !=
that.getRGB(col, row)) return false;
        return true;
    }

    /**
     * Returns a string representation of this
picture.
     * The result is a <code>width</code>-by-
<code>height</code> matrix of pixels,
     * where the color of a pixel is represented using
6 hex digits to encode
     * the red, green, and blue components.
     *
     * @return a string representation of this picture
     */
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append(width + "-by-" + height + " picture
(RGB values given in hex)\n");
        for (int row = 0; row < height; row++) {
            for (int col = 0; col < width; col++) {
                int rgb = 0;
                if (isOriginUpperLeft) rgb =
image.getRGB(col, row);
                else rgb = image.getRGB(col, height -
row - 1);
                sb.append(String.format("#%06X ", rgb
& 0xFFFFFF));
            }
            sb.append("\n");
        }
        return sb.toString().trim();
    }

    /**
```

```
     * This operation is not supported because
pictures are mutable.
     *
     * @return does not return a value
     * @throws UnsupportedOperationException if called
     */
    public int hashCode() {
        throw new
UnsupportedOperationException("hashCode() is not
supported because pictures are mutable");
    }

    /**
     * Saves the picture to a file in either PNG or
JPEG format.
     * The filetype extension must be either .png or
.jpg.
     *
     * @param name the name of the file
     * @throws IllegalArgumentException if {@code
name} is {@code null}
     */
    public void save(String name) {
        if (name == null) throw new
IllegalArgumentException("argument to save() is
null");
        if (name.length() == 0) throw new
IllegalArgumentException("argument to save() is the
empty string");
        File file = new File(name);
        if (file == null) throw new
IllegalArgumentException("could not open file: '" +
name + "'");
        filename = file.getName();
        String suffix =
filename.substring(filename.lastIndexOf('.') + 1);
        if ("jpg".equalsIgnoreCase(suffix) ||
"png".equalsIgnoreCase(suffix)) {
            try {
                ImageIO.write(image, suffix, file);
            } catch (IOException e) {
                e.printStackTrace();
            }
        } else {
            System.out.println("Error: filename must
end in '.jpg' or '.png'");
```

```
        }
    }

    /**
     * Saves the picture to a file in a PNG or JPEG
image format.
     *
     * @param file the file
     * @throws IllegalArgumentException if {@code
file} is {@code null}
     */
    public void save(File file) {
        if (file == null) throw new
IllegalArgumentException("argument to save() is
null");
        filename = file.getName();
        if (frame != null) frame.setTitle(filename);
        String suffix =
filename.substring(filename.lastIndexOf('.') + 1);
        if ("jpg".equalsIgnoreCase(suffix) ||
"png".equalsIgnoreCase(suffix)) {
            try {
                ImageIO.write(image, suffix, file);
            } catch (IOException e) {
                e.printStackTrace();
            }
        } else {
            System.out.println("Error: filename must
end in .jpg or .png");
        }
    }

    /**
     * Opens a save dialog box when the user selects
"Save As" from the menu.
     */
    @Override
    public void actionPerformed(ActionEvent e) {
        FileDialog chooser = new FileDialog(frame,
                "Use a .png or .jpg extension",
FileDialog.SAVE);
        chooser.setVisible(true);
        if (chooser.getFile() != null) {
            save(chooser.getDirectory() +
File.separator + chooser.getFile());
        }
```

```
    }

    /**
     * Unit tests this {@code Picture} data type.
     * Reads a picture specified by the command-line
argument,
     * and shows it in a window on the screen.
     *
     * @param args the command-line arguments
     */
    public static void main(String[] args) {
        Picture picture = new Picture(args[0]);
        System.out.printf("%d-by-%d\n",
picture.width(), picture.height());
        picture.show();
    }

}
```