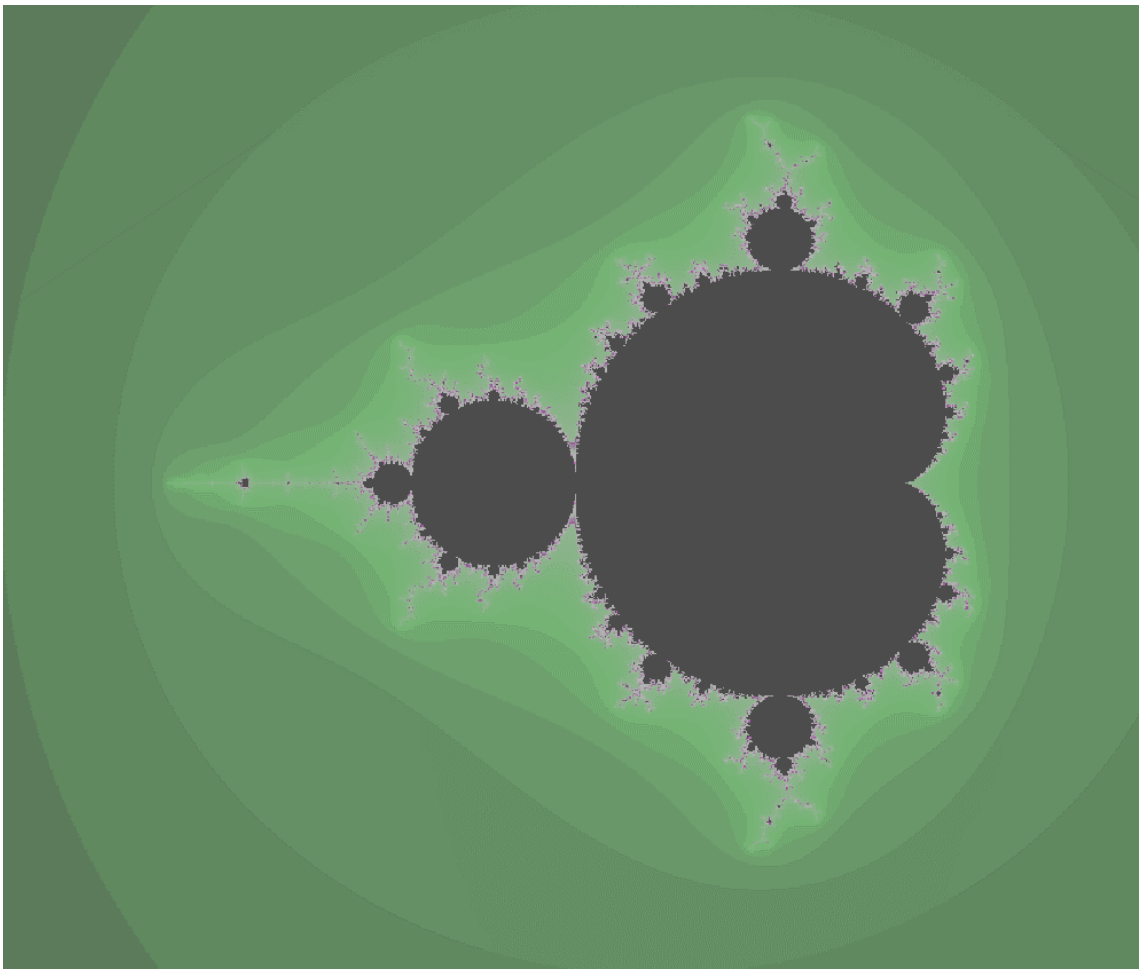
Nolix

Nolix fractals

2020-06-05



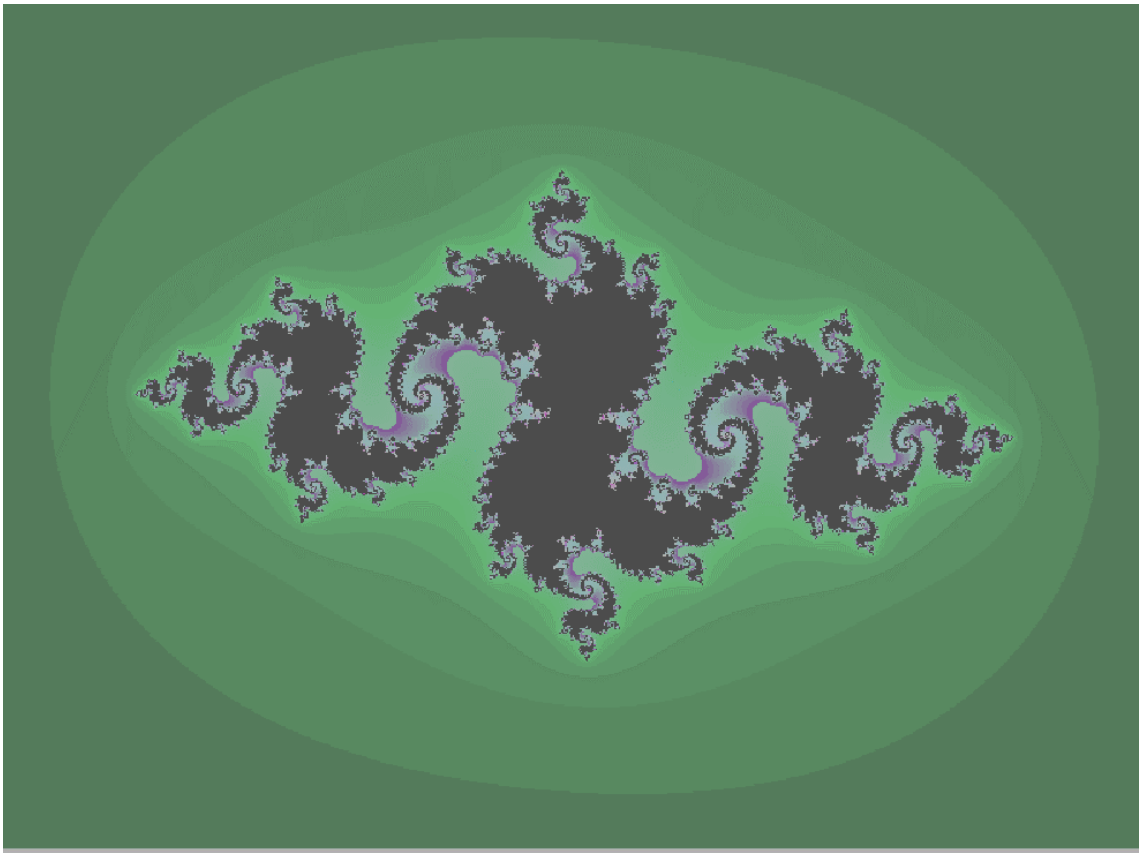


Table of contents

1 Introduction

1.1 What Nolix fractals are

A Nolix fractal is a definition of a specific fractal. A Nolix fractal can generate an image that visualizes the fractal. So, a Nolix fractal is not an image, it is a **definition** for a **unique** fractal image.

1.2 Why to use Nolix fractals

- The parameters of a Nolix fractals are very **general**. Any fractal function, coordination system section, zoom factor, number of iterations, decimal number precision or coloring definition can be chosen.
- Nolix fractals can be calculated using **multi-threading**. This makes the generation of fractal images much faster.

1.3 Where the Nolix fractals are

Nolix fractals are **declared** in the **GenericMathAPI** which is in the Nolix library. The Nolix library contains also an **implementation** of the GenericMathAPI.

1.4 Structure of this document

The mathematical context for Nolix fractals are described in *chapter 2*. *Chapter 3* shows how Nolix fractals can be used in code. In c*hapter 4* there are shown some nice examples of fractals.

2 Mathematical Context

2.1 Motivation

This chapter describes the principle how Nolix fractals work. This chapter explains **all parameters** of a Nolix fractal.

There are **different** ways to create fractals. Nolix fractals are defined by **rows of complex numbers**.

For this chapter, you need to know …

- … what complex numbers are and how calculations with complex numbers are done.
- … what mathematical rows are.

2.2 Theory

For a Nolix fractal there is given a complex row $(a_n(c)): \mathbb{N} \longrightarrow \mathbb{C}$, whereas c is a complex number. We call $a_n(c)$ a **parametrized complex row**.

Example (parametrized complex row)

$(a_1(c)) \coloneqq 0$

$(a_n(c)) \coloneqq {a_{n-1}}^2 + c$

$\boldsymbol{n}$	$\boldsymbol{a_n(0)}$	$\boldsymbol{a_n(1)}$	$\boldsymbol{a_n(\mathrm{i})}$	$\boldsymbol{a_n(1+\mathrm{i})}$
1	0	0	0	0
2	0	1	i	$1+\mathrm{i}$
3	0	2	$-1+\mathrm{i}$	$1+3\mathrm{i}$
4	0	5	$-\mathrm{i}$	$-7+7\mathrm{i}$
5	0	26	$-1+\mathrm{i}$	$1+97\mathrm{i}$
10	0	...	$-\mathrm{i}$	...
100	0	...	$-\mathrm{i}$	...
1000	0	...	$-\mathrm{i}$	...

We see that:

- $a_n(0) = 0$ for all n
- $a_1(c) = 0$ for all c
- $a_2(c) = c$ for all c

Definitions

For a Nolix fractal is also given a so-called **maximum magnitude**, which is a real number. Farther, for a Nolix fractal is given a so-called **maximum iteration count**, which is a natural number.

So far, a Nolix fractal consists of the following things.

- a parametrized complex row $(a_n)(c)$
- a maximum magnitude M whereas $M \in \mathbb{R}$
- a maximum iteration count N whereas $N \in \mathbb{N}$

Motivation (convergence grade)

For painting a Nolix fractal, we take a 2-dimensional coordination system. We interpret a point (x, y) in the coordination system as the complex number $x + y\mathrm{i}$. Note that x and y can be any decimal number or real number and do need to be integers.

For a complex number $z = x + y\mathrm{i}$, we will calculate a **convergence grade**. We use an own, suitable definition for a convergence grade and we know there exist other definitions.

Definition (convergence grade)

- The convergence grade of a complex number z is the smallest natural number n with $|a_n(z)| \leq M$ **if** such an n exists **and if** $n \leq N$.
- …Otherwise the convergence grade is N.

Definition in other words (convergence grade)

g is called convergence grade of $z \in \mathbb{C}$.

$$g := \begin{cases} \min(\min(n||a_n| \leq M), N) & if\ exists \\ N & else \end{cases}$$

Painting fractals

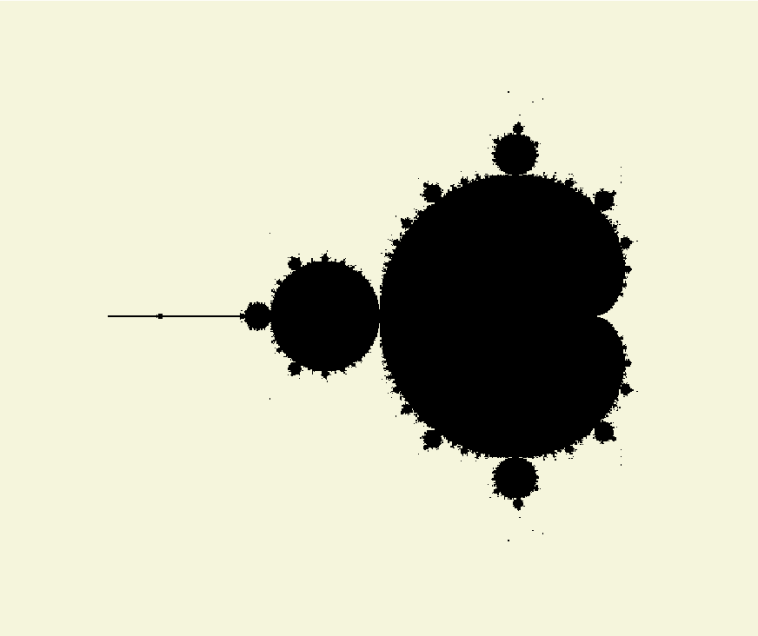
When we have calculated the convergence grades for all selected points resp. for their represented complex numbers, we assign colors to the convergence grades.

- E.g. a convergence grade g gets the Color black if $g = M$ and the Color white otherwise.
- E.g. a convergence grade gets a darker color the bigger the convergence grade is.

Now, in a chosen section on the coordination system, we paint each point in the color of its convergence grade.

At this point, the principal theory is complete. For a suitable parametrized complex row, maximum magnitude and maximum iteration count, we get wonderful fractal images, if we calculate the convergence grade for a certain set of points and if we assign different colors to the convergence grades.

Example (Bicolored Mandelbrot fractal)



parametrized complex row	$a_1(c) \coloneqq 0 \quad a_n(c) \coloneqq {a_{n-1}}^2 + c$
maximum magnitude	2.5
maximum iteration count	100
color function	$g \mapsto \begin{cases} \text{Black} & if\ g = 100 \\ \text{Beige} & else \end{cases}$
coordination system section	$\left\{(x,y) \middle\vert \begin{matrix} x \in (-2.5,\ -2.49,\ \dots\ , 1.48, 1.49), \\ y \in (-1.5, -1.49, \dots, 1.48, 1.49) \end{matrix} \right\}$

About Mandelbrot fractals

A fractal that is defined by a row $(a_n)(c)$ with $a_n(c) \coloneqq {a_{n-1}}^2 + c$ is called **Mandelbrot fractal**. The Mandelbrot fractal is a very popular fractal.

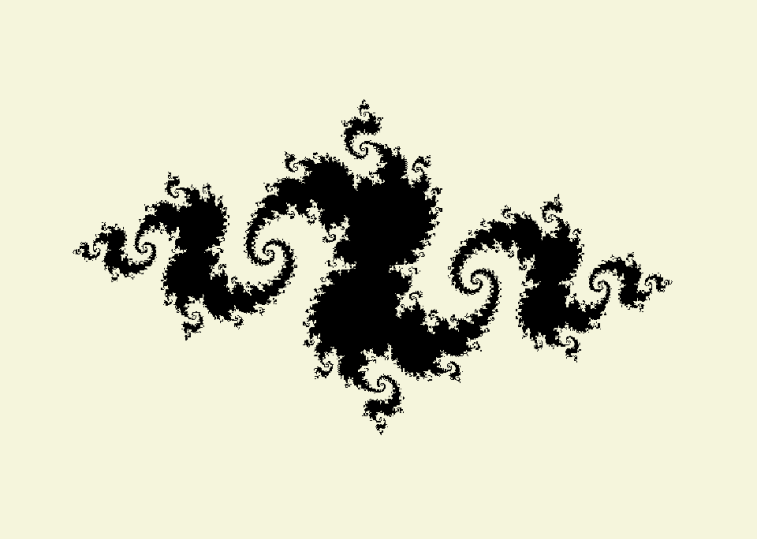
When the row $(a_n)(c)$ with $a_n = 0 \quad a_n(c) \coloneqq {a_{n-1}}^2 + c$ is given, then the **Mandelbrot set** is:

$$\{z \in \mathbb{C} | \exists N \in \mathbb{N}: \forall n \in \mathbb{N}: |a_n(z)| < N\}$$

The example above does not exactly paint the Mandelbrot set. The more precisely the fractal would be calculated, the smaller is the difference between the black area of the fractal and the Mandelbrot set.

Note that images of fractals can show some characteristic properties, but they are not mathematical proofs.

Example (Bicolored Julia fractal)



parametrized complex row	$a_1(c) \coloneqq c \quad a_n(c) \coloneqq {a_{n-1}}^2 - 0.8 + 0.15\mathrm{i}$
maximum magnitude	2.5
maximum iteration count	100
color function	$g \mapsto \begin{cases} \text{Black} & if\ g = 100 \\ \text{Beige} & else \end{cases}$
coordination system section	$\left\{(x,y) \middle\vert \begin{matrix} x \in (-2.0,\ -1.99,\ \dots, 1.99, 2.00), \\ y \in (-1.5, -1.49, \dots, 1.48, 1.49) \end{matrix} \right\}$

About Julia fractals

A fractal that is defined by a row $(a_n)(c)$ with $a_1(c) = c$ and $a_n(c) \coloneqq {a_{n-1}}^2 + j$ whereas $j \in \mathbb{C}$ is called **Julia fractal**. j is called **Julia constant** of the Julia fractal.

3 Fractals

3.1 About the GenericMathAPI

The GenericMathAPI is an API which declares the Nolix fractals. The GenericMathAPI consists of the package ‘ch.nolix.techAPI.genericMathAPI’.

For painting fractals, we will use the following classes from the GenericMathAPI.

Interface	Use
IFractal	Represents a fractal.
IFractalBuilder	Can build IFractals.
IImageBuilder	Provides a fractal image that is generated in real-time.
IComplexNumber	Represents a complex number.
IComplexNumberFactory	Can build IComplexNumbers.
IClosedInterval	Represents a real closed interval.
IClosedIntervalFactory	Can create IClosedIntervals.

3.2 Register an implementation for the GenericMathAPI

```
import ch.nolix.tech.genericMath.GenericMathRegistrator;

…

GenericMathRegistrator.register();
```

The static method ‘register’ of the GenericMathRegistrator registers an implementation for the complete GenericMathAPI at the ClassProvider.

The GenericMathRegistrator class is in the package ‘ch.nolix.tech.genericMath’.

3.3 Create a IFractalBuilder

```
import ch.nolix.core.classProvider.ClassProvider;

…

IFractalBuilder fractalBuilder =
ClassProvider.create(IFractalBuilder.class);
```

When an instance of an interface is needed, the static method ‘create’ of the ClassProvider can create it, when an implementation for the interface was registered. The method ‘create’ takes the interface of the desired instance as parameter.

The ClassProvider class is in the package ‘ch.nolix.core.classProvider’.

3.4 Create a IFractal from a IFractalBuilder

```
IFractalBuilder fractalBuilder;

…

IFractal fractal = fractalBuilder.build();
```

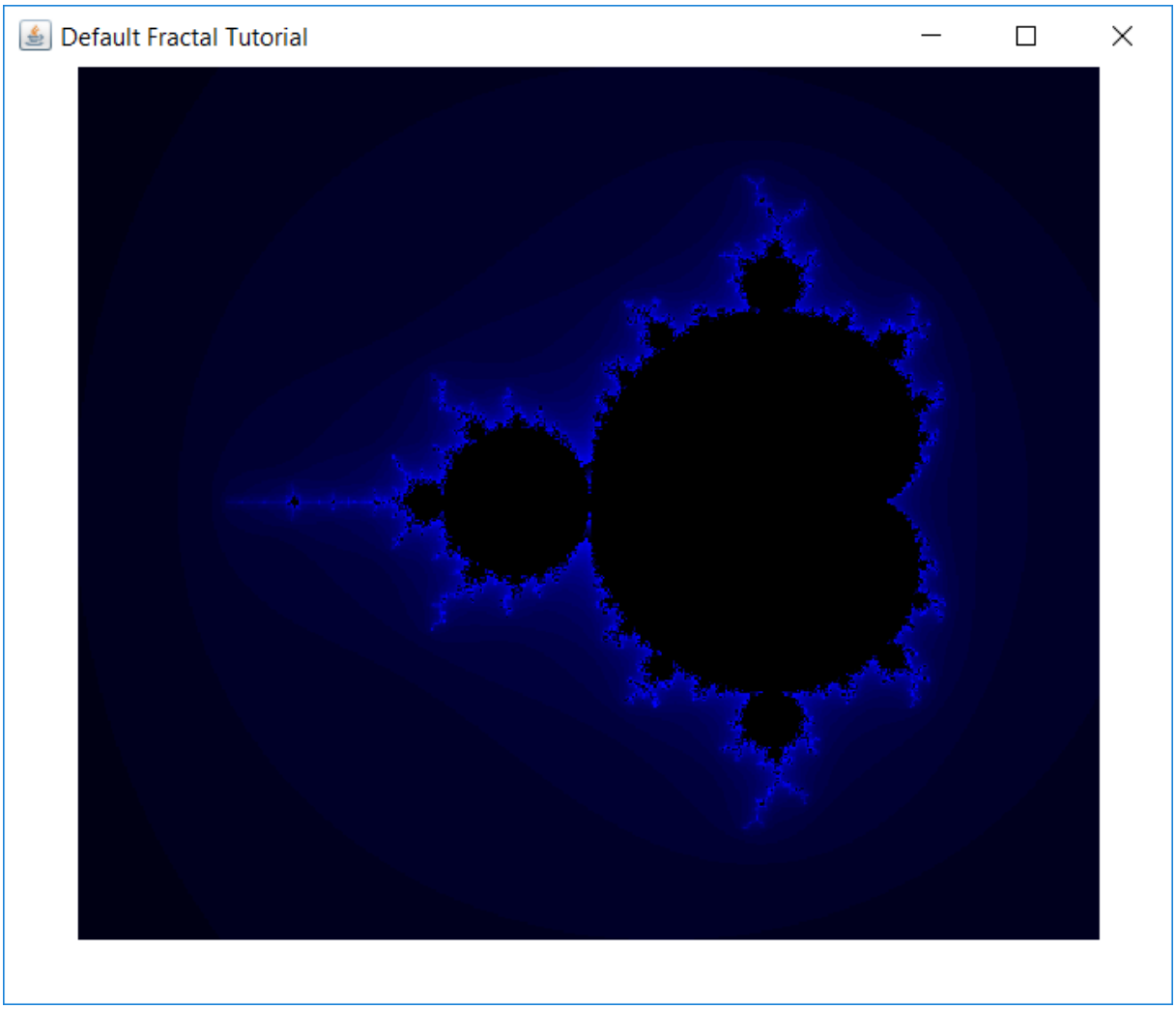
3.5 Create an image from a IFractal

```
import ch.nolix.element.image.Image;

…

IFractal fractal;

…

Image image = fractal.toImage();
```

The Image class is in the package ‘ch.nolix.element.image’.

3.6 Show an image from a IFractal on the screen

```
import ch.nolix.element.GUI.Frame;
import ch.nolix.element.widgets.ImageWidget;

IFractalBuilder fractalBuilder =
ClassProvider.create(IFractalBuilder.class);

IFractal fractal = fractalBuilder.build();

Image image = fractal.toImage();

new Frame("Default Frame Tutorial", new ImageWidget(image));
```

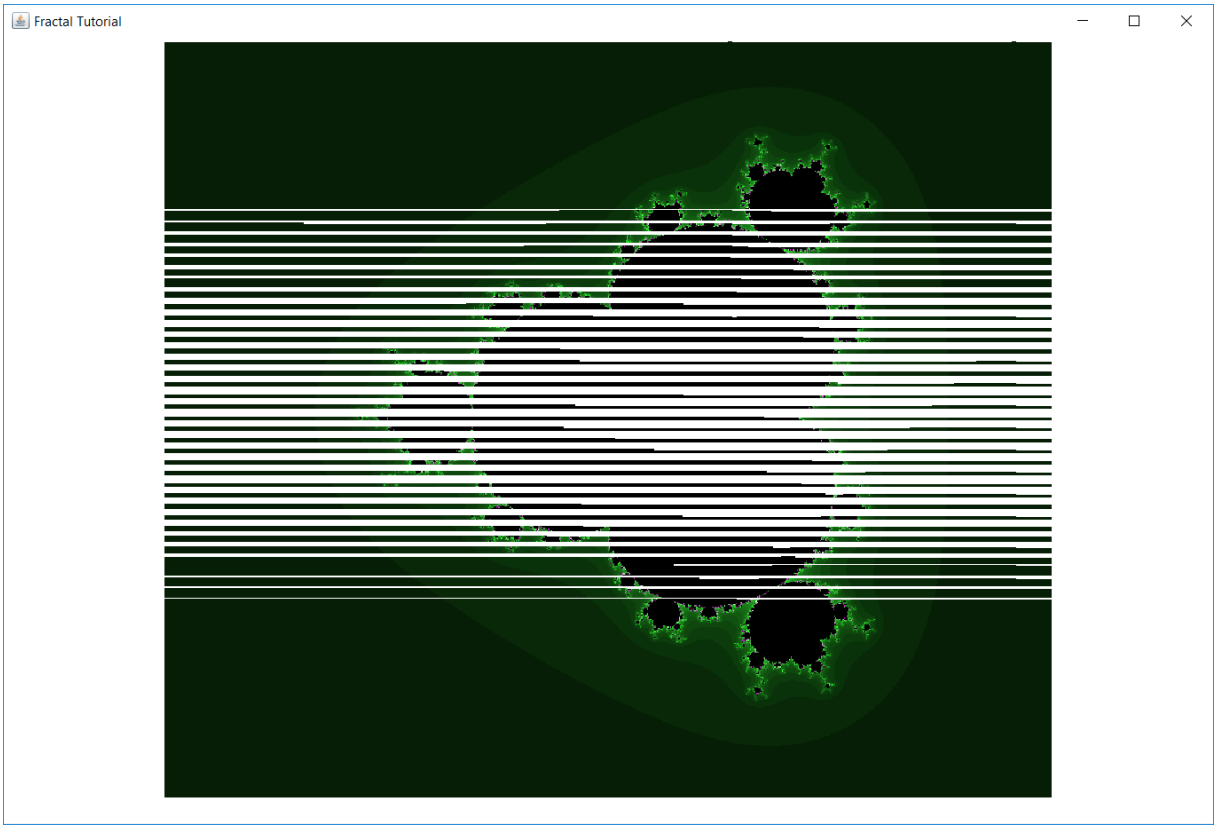


The Fractals that are created by a IFractalBuilder with default settings are Mandelbrot fractals.

The Frame class is in the package ‘ch.nolix.element.GUI. The ImageWidget class is in the package ‘ch.nolix.element.widgets.

3.7 Show the generation of an image from a IFractal on the screen

```
import ch.nolix.core.sequencer.Sequencer;

IFractal fractal;

…

ImageBuilder imageBuilder = fractal.startImageBuild();
Image image = imageBuilder.getImage();
Frame frame = new Frame(“Fractal Tutorial”, new ImageWidget(image));

Sequencer
.asLongAs(() -> frame.isAlive())
.afterAllMilliseconds(100)
.run(() -> frame.refresh());
```



To see an effect, there has to be chosen a fractal that takes enough time to generate an image. It is necessary to update the frame to see the changes on the image.

The Sequencer class is in the package ‘ch.nolix.core.sequencer’.

3.8 Set the scale of the BigDeciamls of a Fractal

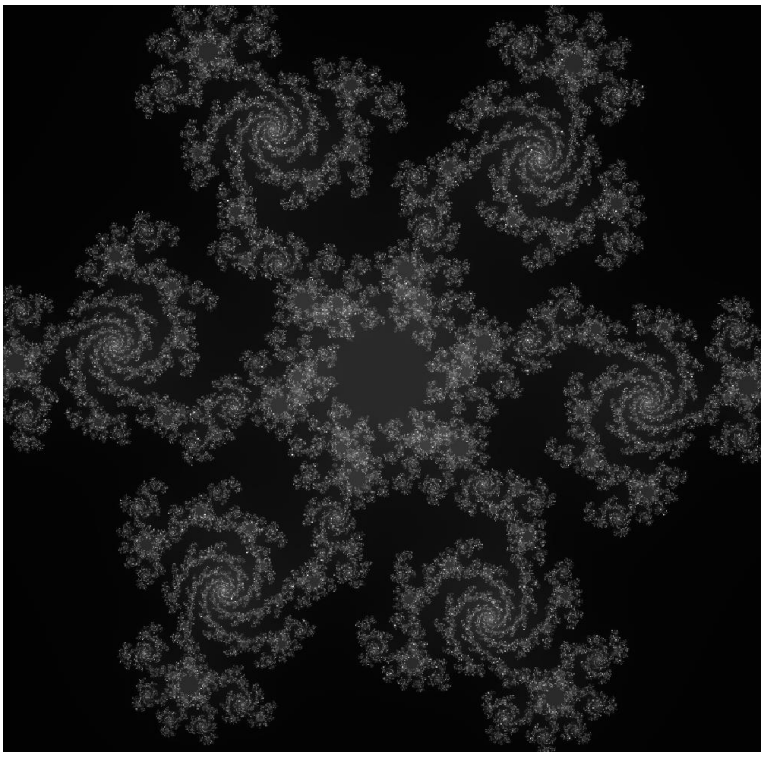
```
IFractalBuilder fractalBuilder;

…

fractalBuilder.setBigDecimalScale(20);
```

All numbers a Fractal calculates are BigDecimals. The scale of a BigDecimal is the precision of the BigDecimal, that is the number of decimal places. All numbers of a Fractal will have the same scale, that means the same precision. The default BigDecimal scale for Fractals is 10.

The bigger the scale of the BigDecimals of a Fractal is, the bigger is the precision of the calculations. But also the bigger the costs are for the calculations. A high BigDecimal scale may be needed for showing a small section of a Fractal in a big picture. Also a big iteration count makes sense only if the BigDecimal scale is big enough.

4 Examples

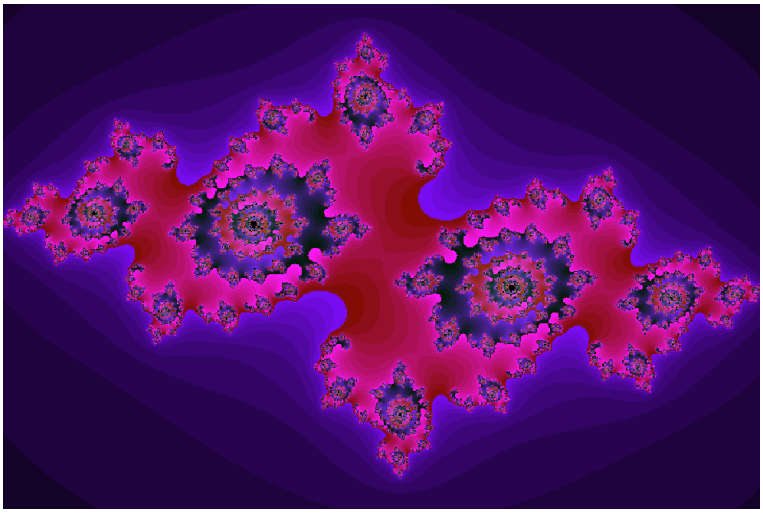
4.1 Example 1



parametrized complex row	$a_1(c) \coloneqq c \quad a_n(c) \coloneqq {a_{n-1}}^6 - 0.8 + 0.15\mathrm{i}$
maximum magnitude	2.5
maximum iteration count	500
color function	$g \mapsto \begin{cases} \text{Black} & if\ g = 500 \\ RGB\ with\ R,G,B = g \bmod 256 & else \end{cases}$
coordination system section	$\left\{(x,y)\middle\vert \begin{matrix} x \in (-1.0,\ -0.9975,\ \dots,0.9975,1.0), \\ y \in (-1.0,-0.995,\dots,0.9977,1.0) \end{matrix}\right\}$
BigDecimal scale	20

```
IFractalBuilder fractalBuilder;

…

var maxIterationCount = 500;

var j =
ClassProvider.create(IComplexNumberFactory.class).create(-0.8, 0.15);

fractalBuilder
.setRealComponentInterval(-1.0, 1.0)
.setImaginaryComponentInterval(-1.0, 1.0)
.setWidthInPixel(800)
.setSequencesStartValuesFunction(c -> new IComplexNumber[]{c})
.setSequencesNextValueFunctionFor1Predecessor(
  (p, c) -> p.getPower(6).getSum(j)
)
.setSequencesMinDivergenceMagnitude(2.5)
.setSequencesMaxIterationCount(maxIterationCount)
.setColorFunction(
  i ->
  i < maxIterationCount ?
  new Color(i % 256, i % 256, i % 256) :
  Color.BLACK
)
.setBigDecimalScale(20);
```

4.2 Example 2



parametrized complex row	$a_1(c) \coloneqq c \quad a_n(c) \coloneqq {a_{n-1}}^2 - 0.745 + 0.25\mathrm{i}$
maximum magnitude	2.5
maximum iteration count	100
color function	$g \mapsto \begin{cases} \text{Black} & if\ g = 100 \\ RGB\ with\ \begin{matrix} R = 10g \bmod 256 \\ G = g \bmod 256 \\ B = 20g \bmod 256 \end{matrix} & else \end{cases}$
coordination system section	$\left\{(x,y) \middle\vert \begin{matrix} x \in (-1.5,\ -1.49625,\ \dots, 1.49625, 1.5), \\ y \in (-1.0, -0.9975, \dots, 0.9975, 1.0) \end{matrix} \right\}$
BigDecimal scale	20

```
IFractalBuilder fractalBuilder;

…

var maxIterationCount = 100;

var j =
ClassProvider.create(IComplexNumberFactory.class).create(-0.745, 0.25);

fractalBuilder
.setRealComponentInterval(-1.5, 1.5)
.setImaginaryComponentInterval(-1.0, 1.0)
.setWidthInPixel(800)
.setSequencesStartValuesFunction(c -> new IComplexNumber[]{c})
.setSequencesNextValueFunctionFor1Predecessor(
  (p, c) -> p.getSquare().getSum(j)
)
.setSequencesMinDivergenceMagnitude(2.5)
.setSequencesMaxIterationCount(maxIterationCount)
.setColorFunction(
  i ->
  i < maxIterationCount ?
  new Color((10 * i) % 256, i % 256, (20 * i) % 256) :
  Color.BLACK
)
.setBigDecimalScale(20);
```