Nolix

Nolix fractals

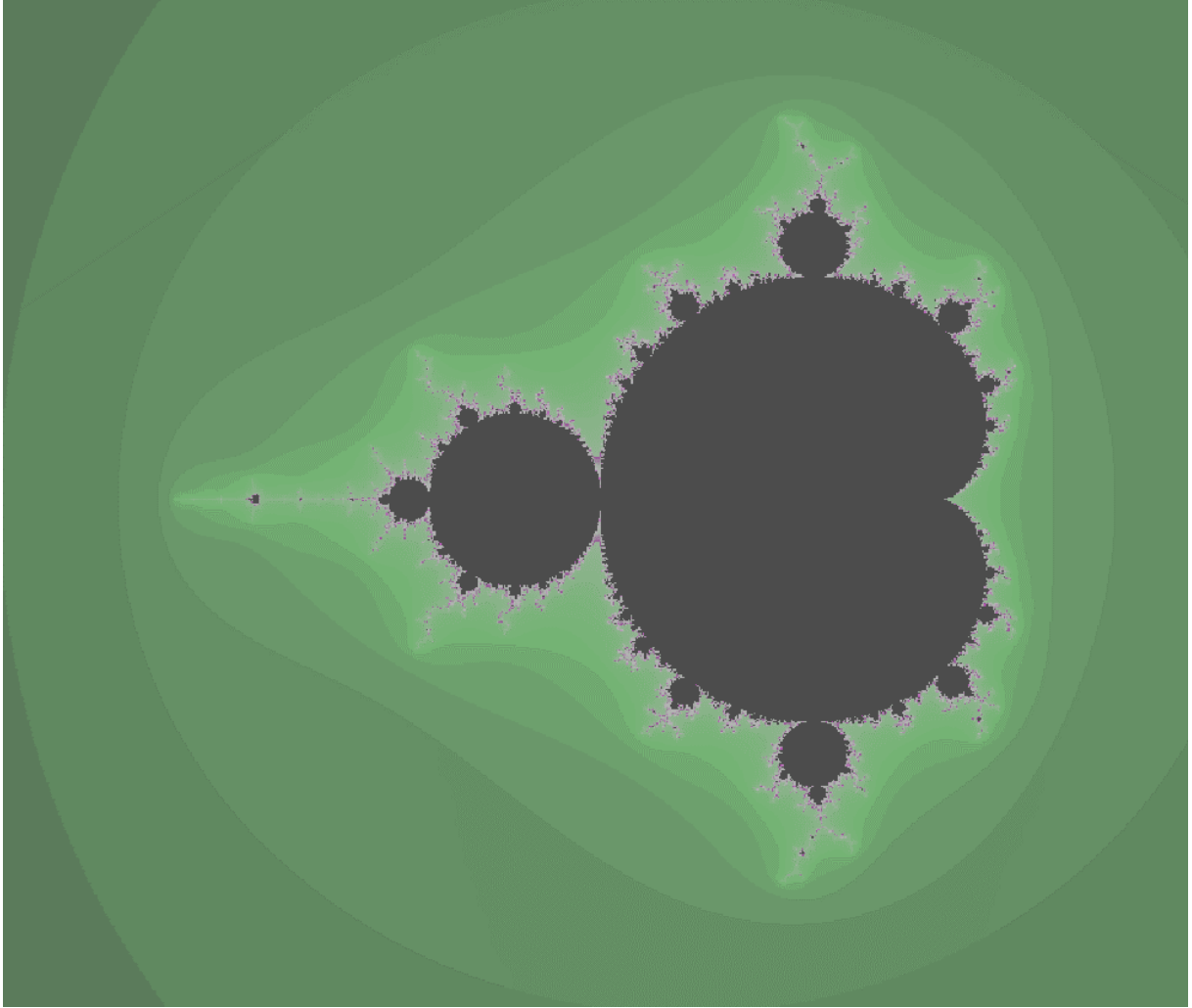## 2019-03-10
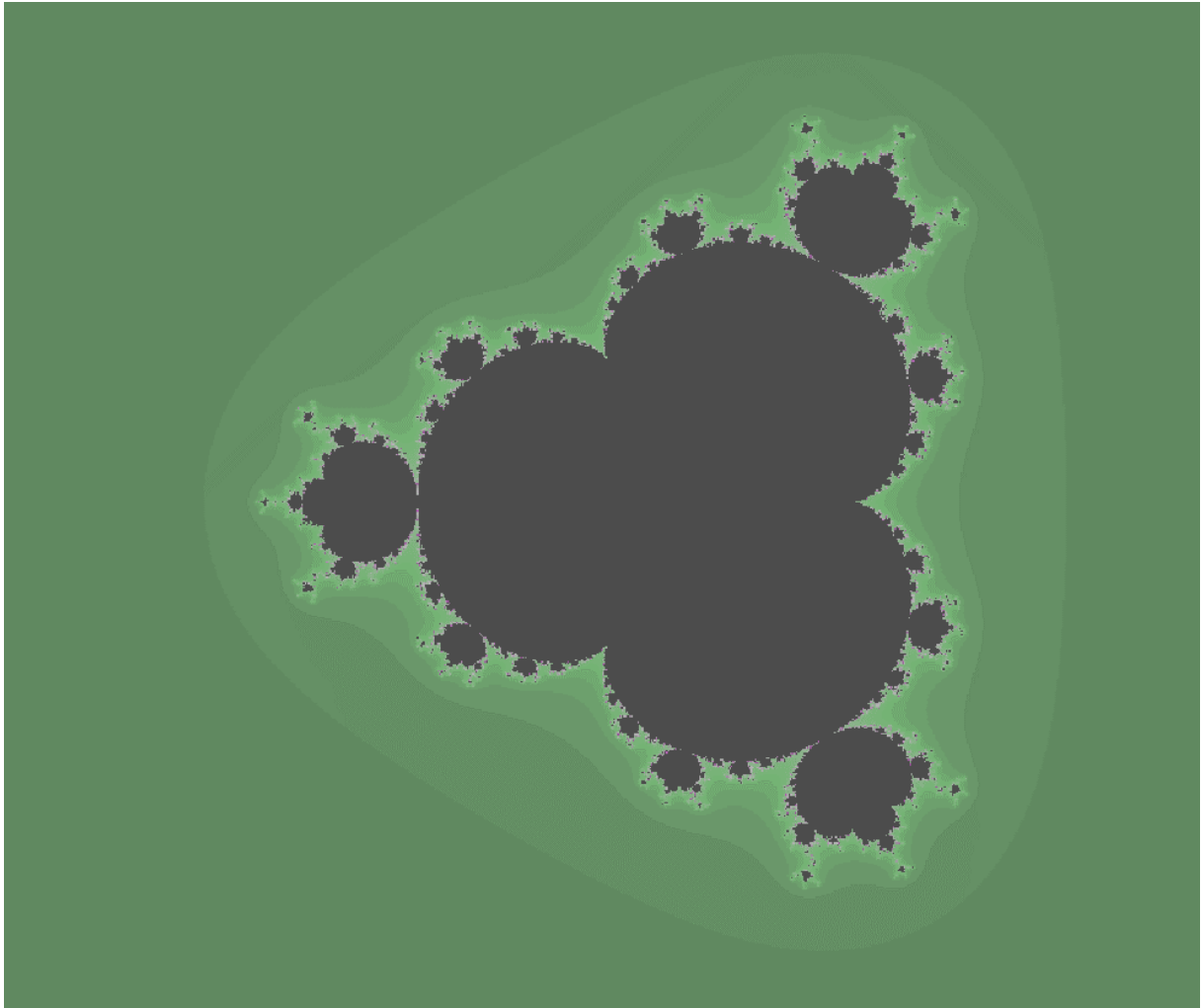
Nolix

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 to create a **unique** fractal image. How Nolix fractals actually work is described in detail in chapter 2.

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.

2 Theory for Nolix fractals

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)

$$(a_1(c)) := 0$$
$$(a_n(c)) := a_{n-1}{}^2 + c$$

n	$a_n(0)$	$a_n(1)$	$a_n(i)$	$a_n(1+i)$
1	0	0	0	0
2	0	1	i	$1+i$
3	0	2	$-1+i$	$1+3i$
4	0	5	$-i$	$-7+7i$
5	0	26	$-1+i$	$1+97i$
10	0	…	$-i$	…
100	0	…	$-i$	…
1000	0	…	$-i$	…

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 $|a_n(z)| \le M$ **if** such an n exists **and if** $n \le N$.
- …Otherwise the convergence grade is N.

Definition in other words (convergence grade)

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

$$g := \begin{cases} \min(\min(n||a_n| \le M), N) \ if \ exists \\ \qquad\qquad N \qquad\qquad\qquad else \end{cases}$$
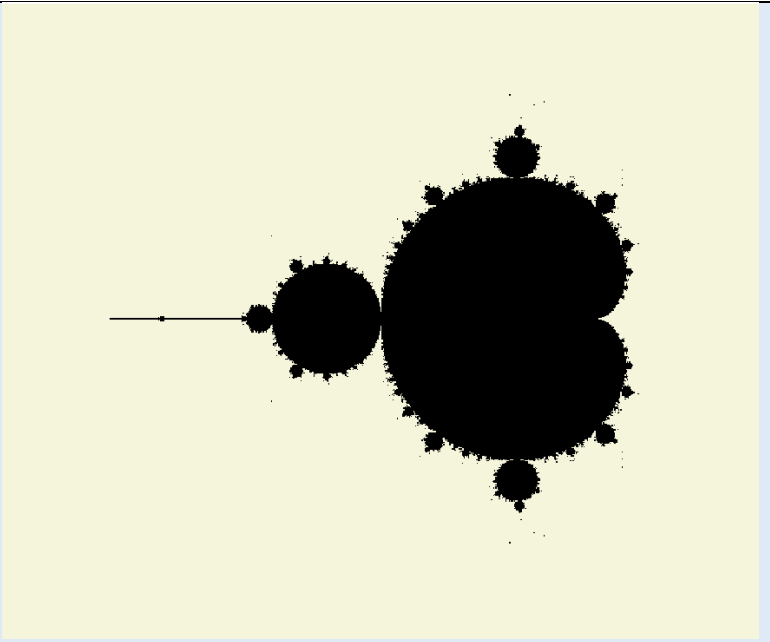
Painting fractals

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 (Black-White Mandelbrot fractal)

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

About Mandelbrot fractals

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

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

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

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.

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 How to 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 is in the package 'ch.nolix.tech.genericMath'.

3.3 How to 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 is in the package 'ch.nolix.core.classProvider'.

3.4 How to create a IFractal from a IFractalBuilder

```
IFractalBuilder fractalBuilder;

…

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

3.5 How to create an image from a IFractal

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

IFractal fractal;

…

Image image = fractal.toImage();
```
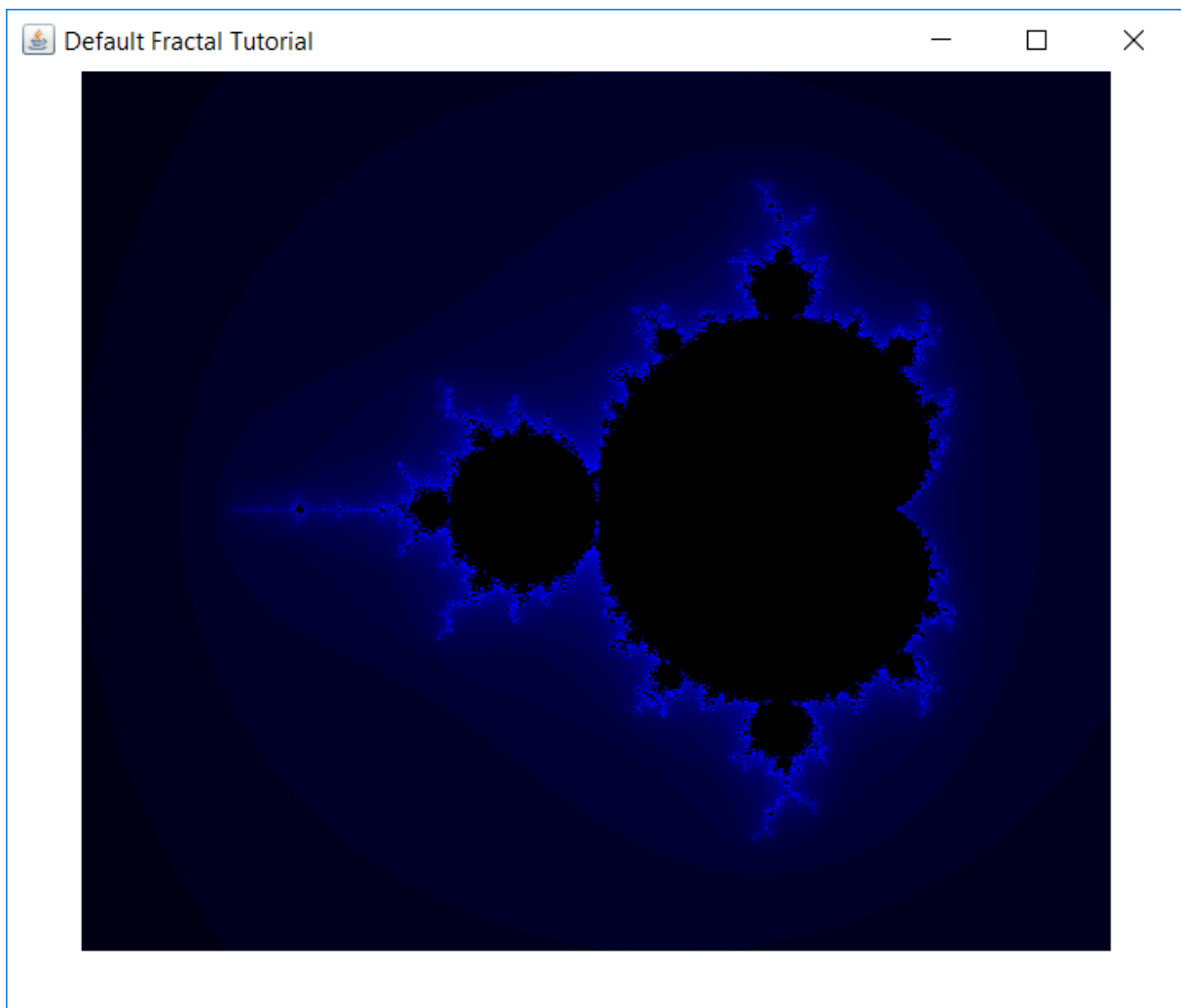
The Image class is in the package 'ch.nolix.element.image'.

3.6 How to show an image from a IFractal on the screen

```
import ch.nolix.element.GUI.Frame;
import ch.nolix.element.GUI.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 FractalBuilder with default settings are Mandelbrot fractals.

The Frame class is in the package 'ch.nolix.element.Frame'.

3.7 How to create an image from a IFractal using multi-threading

```
IFractal fractal;

…

IImageBuilder imageBuilder = fractal.startImageBuild();

Image image = imageBuilder.getRefImage();

boolean finished = imageBuilder.isFinished();
```

The method 'startImageBuilder' of a IFractal **starts the generation** of an image and returns an IImageBuilder. The method 'getRefImage' of an IImageBuilder returns the image that has its final width and length, but can still be in generation. The method 'isFinished' of an IImageBuilder returns true when the image is generated completely.

The method 'startImageBuild' is recommended because more complex fractals can take a longer time to be calculated.

3.8 How to see 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());
```
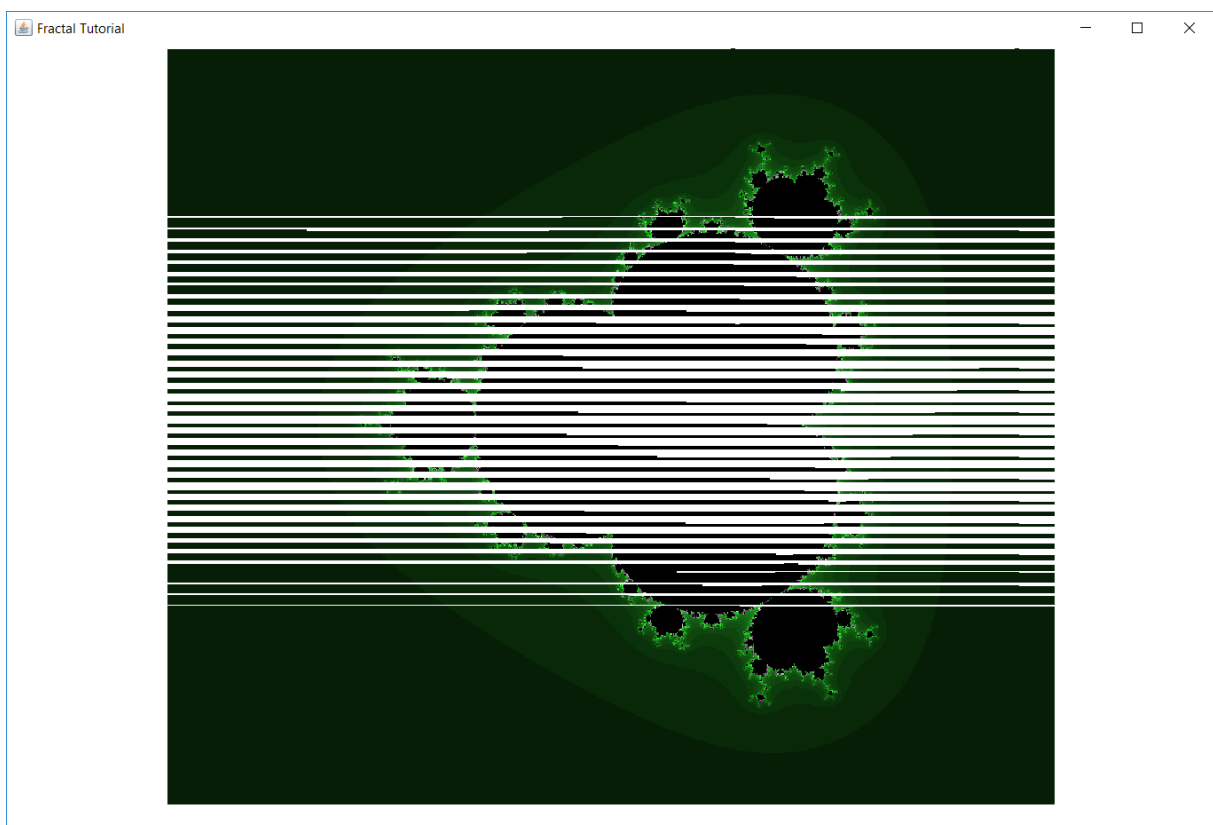

To see an effect, there has to be chosen a fractal that takes enough time to generate an image. It is necessary to update the frame to see the changes on the image.

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