Exercise NMCGJ:
Drawing Fractal Patterns

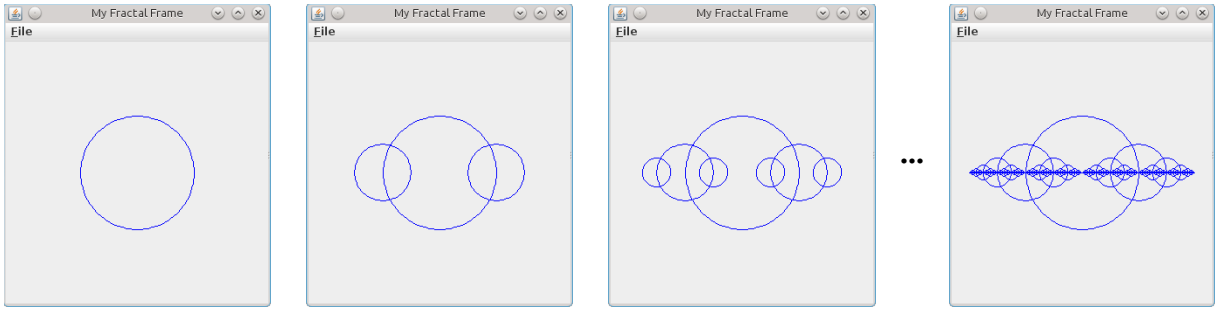
We want to make a program that generates fractal patterns and draws them onto a window.

Fractals

A fractal is a natural phenomenon or a mathematical set exhibiting a pattern that repeats itself at every scale.

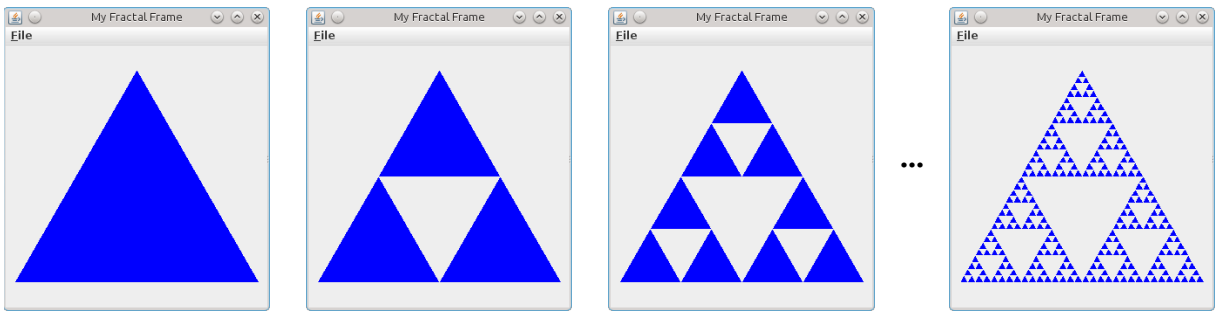
A fractal with circles

The pattern consists of circles along an imaginary line. The procedure is as follows. Draw a circle in the center with a radius of n. Then draw recursively two circles with a radius of $n/2$ whose center points are the intersection points of the previous circle with the imaginary line. The first few steps look like this:



Sierpinski triangle

A Sierpinski triangle can be constructed by starting with an equilateral triangle, and then recursively subdividing it into four smaller equilateral triangles while removing the central subtriangle. It is named after the Polish mathematician Wacław Sierpiński but appeared as a decorative pattern many centuries prior to the work of Sierpiński. The first few steps look like this:



Problem

Make the class `FractalFactory` for generating and drawing the above fractal patterns. The classes `GraphicsFrame` and `GraphicsPanel` are prepared for taking care of the drawing functionality.

The class GraphicsPanel

The class `GraphicsPanel` provides a panel for simple drawings. It inherits from the class `JPanel` which is part of the package `javax.swing`. The following methods are available in the class `GraphicsPanel`:

getWidth()	Returns the current width of the panel.
getHeight()	Returns the current height of the panel.
getDrawColor(): Color	Returns the current color for drawing.
setDrawColor(Color color)	Sets the drawing color.
drawLine(Point point1, Point point2)	Draws a line.
drawPolyline(Point[] points)	Draws a sequence of connected lines.
drawRectangle(Point point1, Point point2, boolean filled)	Draws a (filled) rectangle.
drawOval(Point point1, Point point2, boolean filled)	Draws a (filled) oval, given its bounding box.
drawPolygon(Point[] points, boolean filled)	Draws a (filled) closed polygon.
clear()	Clears the graphics panel.
repaint()	Repaints the graphics panel

The classes `Point` and `Color` are part of the package `java.awt`. They represent a location in integer *(x, y)* coordinates and a color (different formats), respectively. Some predefined colors are `Color.BLACK`, `Color.BLUE`, `Color.RED`, etc.
The packages `java.awt` and `javax.swing` contain classes for creating graphical user interfaces (GUIs) and for painting graphics and images.

The class GraphicsFrame

The class `GraphicsFrame` provides a frame with a graphics panel. It inherits from the class `JFrame` which is part of the package `javax.swing`. The following methods are available in the class `GraphicsFrame`:

start()	Visualizes the frame
close()	Closes the frame.
setMenuVisible(boolean visible)	Indicates whether the menu bar is visible or not.
getGraphicsPanel(): GraphicsPanel	Gives the graphics panel on the frame.
loadGraphicsFile(File file)	Loads image from a file to the graphics panel.
saveGraphicsFile(File file)	Saves image from the graphics panel to a file.

Some simple dialog frames are also available:

showMessageDialog(String msg, String title)	Shows a dialog with a given message.
showInputDialog(String msg, String init): String	Shows a dialog requesting a string.
showInputDialogInt(String msg, int init): int	Shows a dialog requesting an integer.
showInputDialogDouble(String msg, double init): double	Shows a dialog requesting a floating point.

A demo is given in the `main` method.

The class SketchFrame

The class `SketchFrame` is a subclass of the class `GraphicsFrame`, and uses the above functionality to make a simple sketch pad. The drawing is mainly based on the method `drawLine(Point point1, Point point2)` from the class `GraphicsPanel`. Of course, the class `SketchFrame` inherits all functionality from `GraphicsFrame`, and adds the following methods.

setSketchable(boolean sketchable)	Indicates whether sketching is allowed or not by the user.
startSketch(MouseEvent e)	Starts sketching with the mouse.
continueSketch(MouseEvent e)	Continues sketching with the mouse.

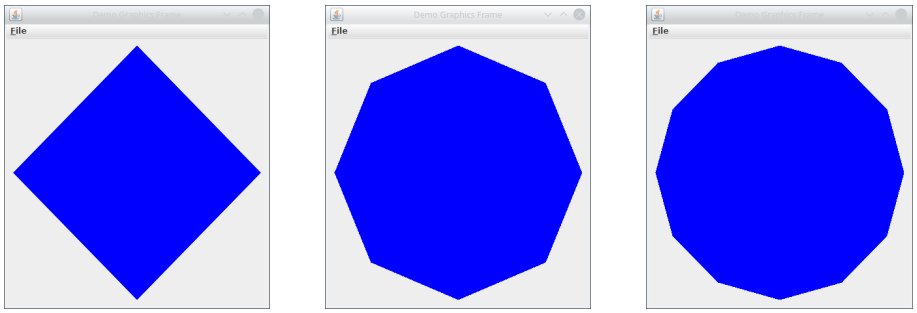
A demo is given in the `main` method.

Implementation of DemoGraphicsFrame

As a preliminary task, we first build a class `DemoGraphicsFrame` that makes a simple polygonal drawing. Make this class as a subclass of the class `GraphicsFrame`, and add a method that draws a regular *n*-gon in blue, centered in the frame:

drawRegularPolygon(int n, boolean filled)	Draws a (filled) regular *n*-gon.

Some examples for $n = 4, 8, 12$:



A possible test for this class could be:

```
// Construct the graphics frame
DemoGraphicsFrame frame = new DemoGraphicsFrame();
frame.start();

// Draw a polygon
frame.drawRegularPolygon(8, true);
```

Implementation of FractalFactory

We are now ready to build the class `FractalFactory` for drawing the desired fractals. So, the following two methods should be provided:

drawCircles(int depth)	Draws the above fractal with circles of a given depth.
drawSierpinskiTriangle(int depth)	Draws the Sierpinski triangle of a given depth.

It is convenient to store the graphics destination (an object of the class `GraphicsPanel`) as a variable of the class `FractalFactory`.

A possible test for this class could be:

```
// Construct the graphics frame
GraphicsFrame frame = new GraphicsFrame("My Fractal Frame");
frame.start();

// Construct the fractal factory
GraphicsPanel graphics = frame.getGraphicsPanel();
graphics.setDrawColor(Color.BLUE);
FractalFactory factory = new FractalFactory(graphics);

// Draw a fractal
factory.drawCircles(5);
```