Object-Oriented Programming

Exercise series 5

Introduction

Starting from this series, some exercises are going to get graphic. In other words, you will write programs that draw pictures. To spare you the effort of going through the Java documentation and learning by yourself how to use the graphical components of the Java library, we provide a sample program `BlackBoard.java` that creates a small window, instantiates the `Graphics2D` class, and shows how to draw a line segment within the window by interacting with the resulting object. The `drawLine()` method invoked by this program takes 4 integer parameters x_1, y_1, x_2, and y_2, that specify the origin (x_1, y_1) and endpoint (x_2, y_2) of the segment to be drawn. Feel free to modify this program in order to experiment and toy around with the graphical facilities offered by the Java library.

Additionally, you might also need to use class methods from the `Math` class, as well as its `PI` constant (a class variable giving an approximation of π as a `double` value). This can be useful if you want to compute the endpoint of a line segment, knowing its origin, its length, and its angle with respect to the horizontal axis. To do so, you can use the `sin()` and `cos()` class methods, both taking an angle expressed in radians (as a `double` value) as parameter.

Exercise 1

Design a `BrokenLine` class able to represent and display a *broken line*, composed of an *origin* (x_0, y_0) and a finite sequence of line segments. This sequence is such that its first segment (if any) begins at (x_0, y_0), and each subsequent segment begins where the previous one ends.

The `BrokenLine` class should implement the following operations :

- Creating a new empty broken line with a given origin (x_0, y_0).
- Appending a new segment to an existing broken line, given its length and its orientation (defined as its angle in radians with respect to an horizontal line).
- Translating a broken line by a given translation vector (δ_x, δ_y), which amounts to applying that translation to the origin and endpoint of each of its line segments.
- Drawing a broken line in a window associated to a given `Graphics2D` object.

You are free to define any auxiliary class required by your solution.

Tips :

- Think carefully about how you will model broken lines. Should segments be instances of a dedicated class ? Should segments maintain the coordinates of their origin and endpoint, or does there exist a better way of representing this information ?
- Recall your solutions to Exercise series 3 and 4. Is there any concept you could reuse here ?

— Test your solution using a program inspired by `BlackBoard.java`. For instance, create and then draw the regular pentagon shown in Figure 1.

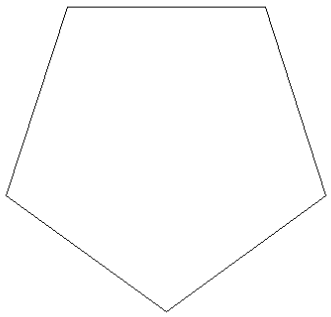

FIGURE 1 – A regular pentagon.

Exercise 2

Expand your solution for Exercise 1 by adding the following operations :

— **Scaling :** Resizing all segments of a broken line by a given real factor to scale up or down the whole figure. Implement this operation in such a way that it leaves the origin of the broken line unaffected.

— **Rotation :** Rotating all segments of a broken line by a given angle (in radians) to tilt the whole broken line around its origin.

Exercise 3

Test your solution to Exercise 2 by replicating one of the figures shown in Figure 2. Each of these figures was obtained by creating a broken line, drawing it once, then rotating it around its origin, drawing it again, and repeating the process until a full rotation has been achieved. Note that, for the top right figure, the broken line was also scaled down after each rotation.

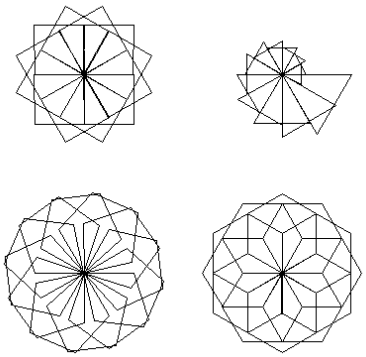

FIGURE 2 – Drawing with four broken lines (square, triangle, pentagon, and hexagon).