Assignment 4

Generic Rubric

- 50% task completion: all requirements implemented
- 20% code correctness: code is correct, demonstrated with unit tests or user tests
- 20% style: code is well-organized, follows coding conventions, and is easy to read
- 10% presentation: code is adequately documented, UML class diagrams are formatted well

Turtle Graphics Editor

In this assignment, you will implement a Turtle graphics editor. Turtle graphics are vector graphics using a relative cursor (the turtle) on a Cartesian plane. The turtle has 3 attributes: a two-dimensional location, a direction, and a pen; it is helpful, to represent the turtle location and direction with `double`. The pen is drawing when it is down, not drawing when it is up.

A drawing is created by moving the turtle with commands that are relative to its position. The turtle traces lines on the drawing.

Drawing Matrix (5 points)

Implement a two-dimensional Matrix data structure to represent a drawing. Every matrix element represents a "pixel" of the drawing; we will use `char` as the Matrix element type to be able to represent different pen effects. If you consider it helpful, you can reuse and extend your Matrix class from Assignment 1.

Drawing Strategy (15 points)

Note that the Matrix drawing has pixels, but the turtle position is in `double`s. We need to convert the turtle traces (lines with end coordinates `x,y` in `double`) into `int` pixel indices for the drawing; we can use Bresenham's Algorithm to determine which pixels to set in the drawing.

Implement the elements of the Strategy design patterns: use the concrete `BresenhamStrategy` class below as a starting point to declare the `Strategy` interface and make the turtle's `Pen` the strategy `Context`. Implement a second strategy following the Naive Algorithm; be careful to avoid division by zero when `x1==x0` (use a conditional statement).

```
public class BresenhamStrategy implements DrawingStrategy {
  @Override
  public void drawLine(Matrix m, double x0, double y0, double x1, double
y1) {
    double dx = Math.abs(x1-x0);
    int sx = x0 < x1 ? 1 : 0;
    double dy = -Math.abs(y1-y0);
    int sy = y0 < y1 ? 1 : -1;
    double error = dx + dy;

    double x = x0;
    double y = y0;
```

```
    while (x != x1 || y != y1) {
      m.setCell((int)x, (int)y, '#');
      double e2 = 2*error;
      if (e2 >= dy) {
        if (x == x1) break;
        error += dy;
        x += sx;
      }
      if (e2 <= dx) {
        if (y == y1) break;
        error += dx;
        y += sy;
      }
    }
  }
}
```

Turtle Commands (30 points)

Implement a read-evaluate-print loop (REPL) for your turtle editor. Use the Command pattern to implement the actions. The REPL should support at least the following commands:

- `quit`: exit the REPL
- `show`: print the current drawing to the command line
- `move x`: move the turtle with the pen up (not drawing) `x` distance in the current direction
- `trace x`: move the turtle with the pen down (drawing) `x` distance in the current direction
- `turn x`: turn the turtle by `x` degrees

Undo Operation (30 points)

Extend your application with an undo/redo functionality. Decide whether undo can be implemented using the Command pattern with reversible commands, or whether it needs the Memento pattern. Depending on your choice, keep track of the history of executed commands/history of snapshots, and on undo, also keep track of undone commands or snapshots to support redo. The redo stack is for "undoing undo": every time that an entirely new command is executed, the redo stack should automatically empty.

Drawing Pattern Library (20 points)

Extend your graphics editor with a library of Composite drawing patterns that combine multiple commands to create a more complex figure. Instantiate the Composite pattern to create at least the following composite commands:

- Rectangle
- The letter `S`
- The letter `E`
- The digit `3`: you can implement it from scratch or as a horizontally flipped `E`
- The text `SE 350`; note that `S` and `5` are the same, the digit `0` is a rectangle.

Below is an example for `SE 350`.

```
###  ###    ###  ###  ###
#    #        #  #    # #
###  ###    ###  ###  # #
  #  #        #    #  # #
###  ###    ###  ###  ###
```

Bonus Question (10 points)

Use the Decorator pattern to optionally decorate the drawing with a frame, as well as with a turtle signature and date.