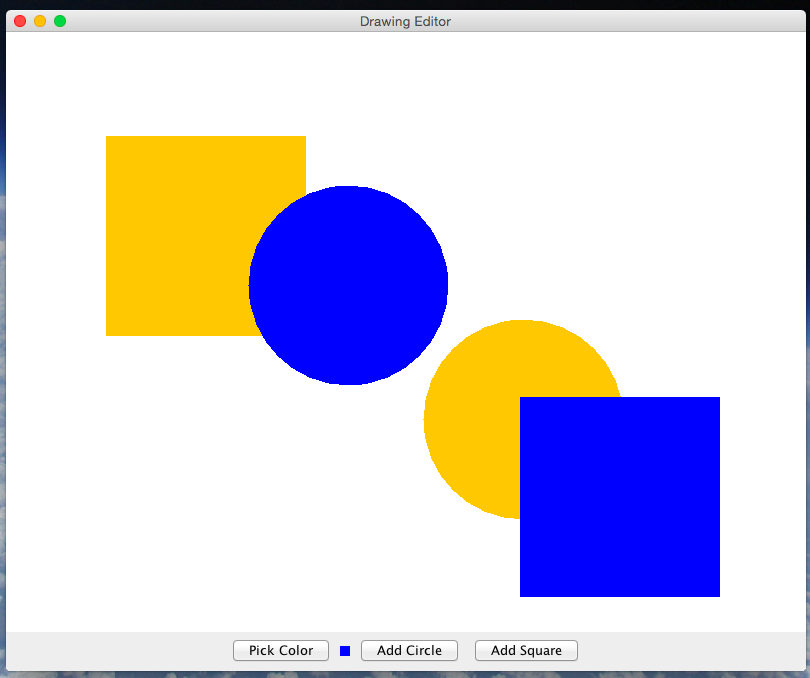
Drawing Editor Lab

Heavily based on “18.6 Case Study and Lab: Drawing Editor” from *Java Methods* by Litvin & Litvin.

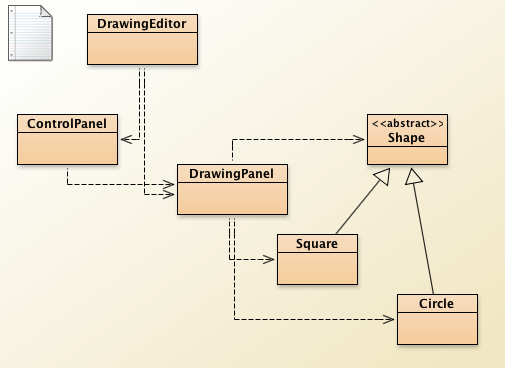
**Introduction**

In this lab we will create a Drawing Editorprogram in which the user can add several filled shapes of different colors and sizes to the picture and drag (*extension: and stretch or squeeze*) them with the mouse (*extension: or cursor keys*). Figure 1 shows a snapshot from the program. The program’s control panel has three buttons: one for choosing a color, another for adding a “circle” to the picture, and a third for adding a “square” to the picture. A small panel displays the currently selected color.

*Figure 1. The DrawingEditor program*

The user can “grab” any shape by clicking on it. The grabbed shape changes from a solid (filled) shape to the outline only. If the user grabs the shape somewhere inside it, then she can drag it with the mouse to a new location (while its size remains unchanged); if she grabs the shape in the vicinity of its border, then she can stretch or squeeze the shape while its center remains in the same location. When another shape is grabbed or the mouse is clicked elsewhere in the canvas, the shape goes back to its solid shape.

*Extension: The last shape added or “grabbed” becomes the “active” shape. The user can move it using cursor (arrow) keys and stretch or squeeze it using the cursor keys with the Shift key held down.*

The program consists of six classes (Figure 2). DrawingEditor (derived from JFrame) represents the program window; ControlPanel (derived from JPanel) represents the panel that holds the buttons; DrawingPanel (derived from JPanel) represents the canvas on which shapes are drawn, andCircle and Square (both derived from Shape) represents shape objects.

*Figure 2. Classes in DrawingEditor*

Our *Drawing Editor*program uses a JColorChooser for choosing a color. The staticshowDialog method is particularly useful.

**Requirements**

* All classes (including instance variables and methods) must be documented using JavaDoc-style comments.

**DrawingEditor**

The DrawingEditor class extends JFrame. In the DrawingEditor’s constructor display “Drawing Editor” in the title bar. Create a DrawingPanel canvas and a ControlPanel controls, passingcanvas to the ControlPanel’s constructor as a parameter (so that controls knows what it controls). Attach canvas and controls to the appropriate regions of the JFrame’s content pane (read about the BorderLayout class for specifics). To complete the DrawingEditor class, provide a standardmain method that creates a new DrawingEditor object which opens a window on the screen.

**ControlPanel**

ControlPanel extends JPanel. Add three buttons to it: “Pick Color,” “Add Circle,” and “Add Square.” Add a JPanel after the “Pick Color” button; its purpose is to show the currently selected color. Callcanvas’s getColor method to obtain the initial color. Attach the appropriate listener to the “Pick Color,” “Add Circle,” and “Add Square” buttons, using the control panel itself as a listener, or, if you prefer, inner action listener classes, or anonymous inline classes. When “Pick Color” is clicked, callcanvas’s pickColor method, then get the selected color back from canvas and show that color on the color display button. When “Add Circle” or “Add Square” is clicked, call canvas’s addCircle oraddSquare method. *Extension: Don’t forget to return the keyboard focus to canvas in either event (read about JComponent’s requestFocusInWindow method).*

*Extension: Support more sophisticated creation of shapes where the user clicks the button for the desired shape and then clicks and drags to specify both where the shape should be centered and what size it should be.*

**DrawingPanel**

This is where most of the work is done. This class is a subclass of JPanel. It can implement theMouseListener and MouseMotionListener interfaces *(extension: and KeyListener)* itself, or, if you prefer, inner classes, or anonymous inline classes. In the DrawingPanel constructor, be sure to add whichever listener you have to this panel as these listeners.

A DrawingPanel maintains a list of shapes (an ArrayList<Shape>) and a reference to the “active shape.” The latter refers to the last added shape or the shape last picked with a mouse (a shape is “picked” when the mouse clicks inside of it or on its border). DrawingPanel should also have anenum field or boolean fields that indicate whether a shape is currently picked and, if so, whether it is being moved or stretched.

DrawingPanel’s constructor and some of the methods are summarized in Table 1. The methods that implement the requirements of the MouseListener, MouseMotionListener, andKeyListener interfaces are not shown. Of these only mousePressed, mouseDragged, andkeyPressed are used.

|  |  |
| --- | --- |
| **Constructor**:  DrawingPanel() | Sets the background color to white and the initial drawing color to a color of your choice. Adds this as the MouseListener, MouseMotionListener, and KeyListener.  Creates an empty shapes list. |
| **Methods**: |  |
| Color getColor() | Returns the current drawing color. |
| Dimension getPreferredSize() | Overrides JCompoent’s getPreferredSize method to return a size large enough to encapsulate a reasonable drawing canvas. |
| void pickColor() | Called from ControlPanel when the “Pick Color” button is clicked. Brings up a JColorChooser and sets the chosen color as the new drawing color. Leaves the drawing color unchanged if the user clicks “Cancel.” |
| void addCircle() | Called from ControlPanel when the "Add Circle" button is clicked. Adds a new circle to the list. The new circle has its center at the center of the drawing panel, a random radius (within a reasonable range) and the current drawing color. The new circle is designated as the “active shape.” |
| void addSquare() | Called from ControlPanel when the "Add Square" button is clicked. Adds a new square to the list. The new square has its center at the center of the drawing panel, a random radius (within a reasonable range) and the current drawing color. The new square is designated as the “active shape.” |
| void paintComponent(Graphics g) | Draws all the shapes in the list. The shapes should be drawn in reverse from the order in which they were added to the list. However, if one of the shapes is “picked,” then this “active shape” should be drawn last, in outline only. |

*Table 1. The constructor and methods of the DrawingPanel class (except the listener methods)*

When the mouse is pressed on shapes, make sure you pick the topmost shape that contains the coordinates of the click. If you add shapes at the end of the list, then you need to scan the list from the end backward to achieve that.

The secret for smooth dragging action is to keep constant the *x*-*y*offsets from the current mouse position to the current shape’s center as the mouse moves. Save the previous mouse position while dragging such that these offsets remain the same as the shape is moved.

*Extension: If the initial click happens in the vicinity of the border of a shape (that is, isOnBorder(x, y) returns true), then instead of dragging the shape make the mouse motion stretch or squeeze it. You need a boolean field (or enum) in DrawingPanel to mark whether you will be moving or stretching the picked shape. The same principle as for moving applies for smooth stretching action, but is a little more complicated.*

*Extension: Enable cursor keys to move the active shape (if not null) or to stretch or squeeze it when the Shift key is held down.*

Don’t forget to call repaint after adding a shape and whenever one of the listener methods changes the appearance of the picture.

**Shape**

This abstract class implements a shape with a given center, radius, and color. Its constructor and methods are summarized in Table 2.

Thoroughly test all the mouse and keyboard action in your program.

|  |  |
| --- | --- |
| **Constructor**:  Shape(Point2D.Double center,    double radius, Color color) | Creates a shape with the center at the specified point and with the specified radius and color. |
| Methods: |  |
| Point2D.Double getCenter() | Returns the the center of the shape. |
| double getRadius() | Returns the radius. |
| void move(double x, double y) | Displaces the center of this shape by (*x*, *y*). |
| void setRadius(double r) | Sets the radius of this shape to *r*. |
| boolean isInside(Point2D.Double point) | Returns true if the point lies inside this shape, false otherwise. (Abstract in Shape class.) |
| *boolean isOnBorder(Point2D.Double point)* | *Extension: Returns true if the point lies approximately on the border of this shape, false otherwise. (Abstract in Shape class.)* |
| void draw(Graphics2D g2, boolean filled) | Draws this shape. Draws a filled shape if filled is true, and a hollow shape otherwise. (Abstract in Shape class.) |

*Table 2. The constructor and public methods of the abstract Shape class*

**Circle and Square**

The Circle and Square classes are derived from the abstract Shape base class and draw their respective shapes.

*Extension: add additional types of shapes.*

**Extensions**

* Support resizing the shapes by clicking near the edge of the shape and dragging to change the size
* Support fine-grained moving of shapes by using the arrow keys to move the active shape
* Support fine-grained re-sizing of shapes by using the arrow keys with the shift key held down to resize the active shape
* Support more than two types of shapes.
* Support more sophisticated creation of shapes where the user clicks the button for the desired shape and then clicks and drags to specify both where the shape should be centered and what size it should be.
* Add more awesome.

**Submission**

* Submit a pull request in GitHub and submit a link to the request with this assignment.