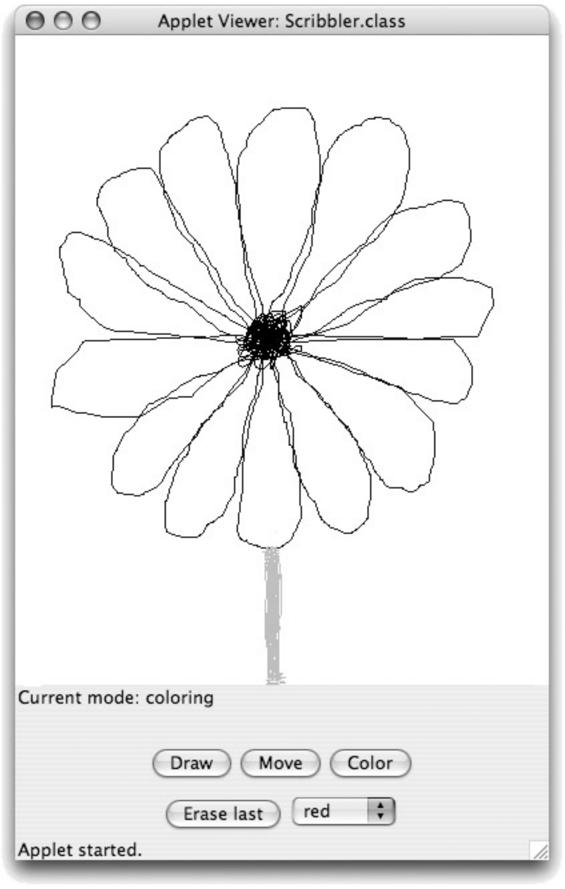
Programming Exercise
Scribbler

Objective: To gain more experience using recursion and recursive data structures.

This week, you will be implementing a simple drawing program we call "Scribbler." A sample of what your program's interface might look like is shown below.



The Scribbler program has three modes: Draw mode, Move mode, and Color mode. The modes behave as follows:

Draw mode: As the user drags the mouse around in the canvas, the program will draw lines following the path of the mouse.

Move mode: In the mode, the user can reposition a scribble by pressing the mouse while touching some part of the scribble and then dragging the scribble to its new location.

Color mode: Clicking on any scribble while in this mode should change its color to that selected in the color menu.

The program starts in Draw mode. Draw mode, Move mode, and Color mode are selected by pressing buttons, and the color used by Color mode is selected by choosing a color from a JComboBox filled with color names.

The program also has an "Erase Last" button that will erase the most recently drawn scribble. Pressing the button repeatedly will erase scribbles in the reverse order of which they were drawn.

How to Proceed We will provide you with a working but incomplete Scribble program as a starter for this week's exercise. It supports only the Draw mode and a simplified Move mode that is capable of moving only the most recently drawn scribble. You will need to add code that will manage your scribbles to allow the various modes and the "Erase last" button to work correctly.

There are a number of step-by-step approaches you could take to complete this program, but it is important that you have a plan, and that you add functionality one step at a time. Here is one possible ordering of the tasks. We recommend that you develop and test your program incrementally – make sure you have a working implementation at each step before moving on to the next.

1. Implement a simplified Color mode. This will be similar to Move mode supported by the starter code, except that you set the color of the most recently drawn scribble instead of moving it. To do this, you will have to add a `setColor` method to the `ScribbleInterface` interface, and the `NonEmptyScribble` and `EmptyScribble` classes.

2. Implement a simplified Erase mode. Here, you respond to the "Erase last" button's `actionPerformed` event by deleting the most recently drawn scribble from the canvas. For now, you will only be able to erase the most recently drawn scribble. A second button press will do nothing.

3. To implement the complete functionality for this program, you will need to keep track of a number of scribbles. You should do this by defining recursive classes to represent collections of scribbles. The interface for these classes will be called `ScribbleCollectionInterface`, and the classes implementing that interface will be named `EmptyScribbleCollection` and `NonEmptyScribbleCollection`.

   Just as we do not know how many line segments will make up a scribble when we start to draw one, we will not know how many scribbles will be drawn and stored in a scribble collection. In your `Scribbler` class you will need a variable with type `ScribbleCollectionInterface` that will be initialized with an object created from `EmptyScribbleCollection`. You will add new scribbles to it using the constructor from `NonEmptyScribbleCollection` as they are drawn. Consider carefully what methods your scribble collection needs to support the functionality of the four modes. We have provided the skeleton of a `ScribbleCollectionInterface` interface, and `NonEmptyScribbleCollection` and `EmptyScribbleCollection` classes with our starter files.

   - Draw mode needs to add a new scribble to the scribble collection after the dragging is done.
   - The "Erase last" button needs to remove the most recently drawn scribble from the canvas and remove it from the scribble collection. This means there is now a new "most recently drawn" scribble, and a second press would remove that one, and so on. Be careful in the case when there are no scribbles left.
   - Move mode and Color mode need to determine if a mouse click takes place on *any* of the scribbles on the canvas, not just the most recently drawn. This requires that your scribble collection classes be able to search through all of the scribbles until either a scribble is located that contains the click point, or it is determined that none of the existing scribbles contain the point. Once you have determined which scribble contains the click point, you can move or color that scribble as you did in the simplified versions of these modes.

Submitting Your Work Before turning in your work, be sure to double check both its logical organization and your style of presentation. Make your code as clear as possible and include appropriate comments describing major sections of code and declarations.

Sketch of classes provided for this program We are providing several classes and interfaces to help you get started. We include here printouts of the main program and the classes and interface for a single scribble. You will need to add additional methods to this code. We have not included the code for the collection interface or collection classes.

```
// A very simple drawing program.
public class Scribbler extends WindowController
                       implements ActionListener {

   // user modes
   // using ints rather than boolean to allow for extension to
   // other modes
   private static final int DRAWING = 1;
   private static final int MOVING = 2;
   private static final int COLORING = 3;

   // empty scribble as placeholder when nothing there.
   private static final EmptyScribble empty = new EmptyScribble();

   // the current scribble
   private ScribbleInterface currentScribble;

   // the collection of scribbles
   private ScribbleCollectionInterface scribbles;

   // stores last point for drawing or dragging
   private Location lastPoint;

   // whether the most recent scribble has been selected for moving
   private boolean draggingScribble;

   // buttons that allow user to select modes
   private JButton setDraw, setMove, setErase, setColor;

   // Choice JButton to select color
   private JComboBox chooseColor;

   // new color for scribble
   private Color newColor;

   // label indicating current mode
   private JLabel modeLabel;

   // the current mode -- drawing mode by default
   private int chosenAction = DRAWING;

   // create and hook up the user interface components
   public void begin() {

      setDraw = new JButton(‘‘Draw’’);
      setMove = new JButton(‘‘Move’’);
      setColor = new JButton(‘‘Color’’);

      JPanel buttonPanel = new JPanel();
      buttonPanel.add(setDraw);
      buttonPanel.add(setMove);
      buttonPanel.add(setColor);

      chooseColor = new JComboBox();
      chooseColor.addItem(``red'');
      chooseColor.addItem(``blue'');
      chooseColor.addItem(``green'');
      chooseColor.addItem(``yellow'');

      setErase = new JButton(``Erase last'');
      JPanel choicePanel = new JPanel();
      choicePanel.add(setErase);
      choicePanel.add(chooseColor);

      JPanel controlPanel = new JPanel(new GridLayout(3,1));
      modeLabel = new JLabel(``Current mode: drawing'');
      controlPanel.add(modeLabel);
      controlPanel.add(buttonPanel);
      controlPanel.add(choicePanel);

      getContentPane().add(controlPanel, BorderLayout.SOUTH);

      // add listeners
      setDraw.addActionListener(this);
      setMove.addActionListener(this);
      setErase.addActionListener(this);
      setColor.addActionListener(this);

      // make the current scribble empty
      currentScribble = empty;

      validate();
   }

   // if in drawing mode then start with empty scribble
   // if in moving mode then prepare to move
   public void onMousePress(Location point) {
      if (chosenAction == DRAWING) {
             // start with an empty scribble for drawing
         currentScribble = empty;

      } else if (chosenAction == MOVING) {
             // check if user clicked on current scribble
         draggingScribble = currentScribble.contains(point);
      }

      // remember point of press for drawing or moving
      lastPoint = point;
   }

   // if in drawing mode, add a new segment to scribble
   // if in moving mode then move it
```

```
   public void onMouseDrag(Location point) {
      if (chosenAction == DRAWING) {
         // add new line segment to current scribble
         Line newSegment = new Line(lastPoint, point, canvas);

         currentScribble =
               new NonEmptyScribble(newSegment, currentScribble);
      } else if (chosenAction == MOVING) {
         if (draggingScribble) {// move current scribble
            currentScribble.move(point.getX() - lastPoint.getX(),
                                 point.getY() - lastPoint.getY());
         }
      }
      // update for next move or draw
      lastPoint = point;
   }

   public void onMouseRelease(Location point) {
      // Add code when have collection of scribbles
   }

   // Set mode according to JButton pressed.
   public void actionPerformed(ActionEvent e) {
      if (e.getSource() == setDraw) {
         chosenAction = DRAWING;
         modeLabel.setText(``Current mode: drawing'');
      } else if (e.getSource() == setMove) {
         chosenAction = MOVING;
         modeLabel.setText(``Current mode: moving'');
      }
   }
}
```

Here is the starting code for `ScribbleInterface`:

```
public interface ScribbleInterface {
  // returns whether point is contained in scribble
  public boolean contains(Location point);

  // move scribble by dx in x-direction and dy in y-direction
  public void move(double dx, double dy);

}
```

Here is the starting code for `NonEmptyScribble`:

```
// A class to represent a non-empty scribble

public class NonEmptyScribble implements ScribbleInterface {
    private Line first; // an edge line of the scribble
    private ScribbleInterface rest; // the rest of the scribble
    public NonEmptyScribble(Line segment, ScribbleInterface theRest) {
        first = segment;
        rest = theRest;
    }

    // returns true iff the scribble contains the point
    public boolean contains(Location point) {
        return (first.contains(point) || rest.contains(point));
    }

    // the scribble is moved xOffset in the x direction
    //    and yOffset in the y direction
    public void move(double xOffset, double yOffset) {
        first.move(xOffset, yOffset);
        rest.move(xOffset, yOffset);
    }
}
```

Here is the starting code for `EmptyScribble`:

```
/* Class representing an empty scribble */
public class EmptyScribble implements ScribbleInterface {
    public EmptyScribble() {}

    // point is never in an empty scribble!
    public boolean contains(Location point) {
        return false;
    }

    // nothing to move, so do nothing!
    public void move(double dx, double dy) {}
}
```