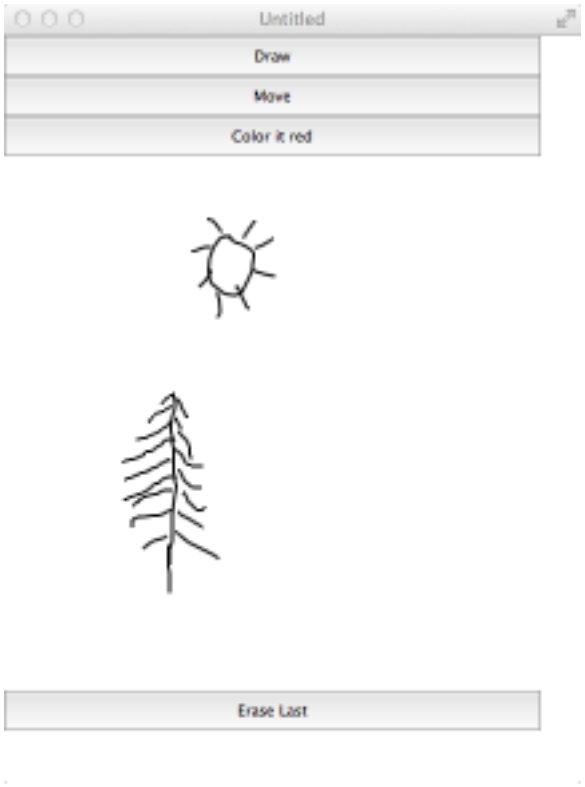
CS 51 Homework Laboratory # 8
Scribbler

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

We're now back with a regular lab, so you will need to come to lab with a design for all of the steps in the "How to Proceed" section. This week, you will be implementing a program we call "Scribbler." You have seen a simpler version of this program in class as a demo.



The Scribbler program has three modes: Draw mode, Move mode, and Color mode. 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 choice component filled with color names. The modes behave as follows:

Draw mode: Drag to draw a new Scribble on the canvas.

Move mode: Move a Scribble by pressing the mouse on it and dragging.

Color mode: Click on any Scribble to change its color to that selected in the color choice component. FIX! ►*Just red for now.*◄

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. Make sure your program does not crash after erasing the last one!

How to Proceed The starter for this lab is a working Scribbler, but it has 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 solve this problem, 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 is similar to Move mode, except that you set the color of the Scribble that contains the mouse point instead of moving it. To do this, you will have to add a color:= method to the Scribble type, and the scribble and emptyScribble classes. Initially, you will only be able to set the color of the most recently drawn Scribble.

2. Implement a simplified Erase mode. Here, you respond to the "Erase last" button's buttonPressed 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. Since you need to keep track of a number of Scribbles, you will need to define an object and class generating a collection of Scribbles. The type of object generated by these classes will be called ScribbleCollection. The base object will be named emptyScribbleCollection and the class `scribbleCollection`. This type, object, and class will be similar to the ball lists in the Chain-Reaction examples in Lecture 21.

   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 our ScribbleCollection. In your Scribbler object/main program you will need a variable with type ScribbleCollection that will be initialized with the object emptyScribbleCollection. You will add new Scribbles to it using the constructor from scribbleCollection as they are drawn. Consider carefully what methods your scribble collection needs to be able to support the functionality of the four modes. We have provided the skeleton of a ScribbleCollection type, scribbleCollection class, and emptyScribbleCollection object.

   - Draw mode needs to add a new Scribble to the ScribbleCollection when the dragging is done.
   - The "Erase last" button needs to remove the most recently drawn Scribble from the canvas and remove it from the ScribbleCollection. 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 ScribbleCollection be able to search through all of its 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.

Sketch of classes provided for this program We are providing some classes and types to help you get started. These are almost identical to the Scribbler examples from lecture. Please download the starter to see the objects, classes, and types we provide before working on your design. Some of these are included at the end of this write-up.

Submitting Your Work The lab is due Monday at 11 PM. When your work is complete you should deposit in the appropriate dropoff folder a folder that contains your program and all of the usual files needed by JBuilder. Make sure the folder name includes your name and the phrase "Lab 8". Also make sure that your name is included in the comment at the top of each Java source file.

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.

Table 1: Grading Guidelines

Value	Feature
	Design preparation (3 points total)
1 points	Simplified color mode
1 points	Erase mode
1 point	`scribbleCollection` class
	Code Quality (5 points total)
2 points	Descriptive comments
1 point	Good names
1 point	Good use of constants
1 point	Appropriate formatting
	Design (6 points total)
1 point	Good use of boolean expressions, loops, conditionals
1 point	Not doing more work than necessary
2 points	Appropriate methods in `scribbleCollection`
2 points	Appropriate recursive structure in `scribbleCollection`
	Correctness (6 points total)
1 point	Drawing the GUI components correctly
1 point	Switching among modes correctly
1 point	Draw mode adds correctly to the `ScribbleCollection`
1 point	Move mode works correctly
1 point	Color mode works correctly
1 point	Erase button works correctly

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. These are very similar to the Scribbler examples from lecture. Here is the code from the basic `Scribbler` class. You may need to add additional methods to it:

```
dialect "objectdrawDialect"
import "Scribble" as sc
import "ScribbleCollection" as scc

type Scribble = sc.Scribble
def emptyScribble = sc.emptyScribble
def scribble = sc.scribble

object {
    inherits graphicApplication.size(400,400)

    // define the button
    def drawButton: Button = button.withLabel("Draw")
    def moveButton: Button = button.withLabel("Move")
    def colorButton: Button = button.withLabel("Color it red")
    def eraseButton: Button = button.withLabel("Erase Last")

    addToNorth(drawButton)
    addToNorth(moveButton)
    addToSouth(eraseButton)
    addToNorth(colorButton)

    // add the button to the top of the canvas

    // add the listener
    drawButton.addButtonListener(self)
    moveButton.addButtonListener(self)
    colorButton.addButtonListener(self)
    eraseButton.addButtonListener(self)

    var lastMouse: Point

    def drawingMode = 1
    def movingMode = 2
    def coloringMode = 3

    var mode: Number := drawingMode // was the mouse pressed on the scribble

    var currentScribble: Scribble

    var dragging: Boolean := true

    // respond to mouse presses on the canvas
    method onMousePress(point:Point) -> Done {
        if (mode == drawingMode) then {
            currentScribble := emptyScribble
        } elseif (mode == movingMode) then {
            dragging := currentScribble.contains(point)
        }
        lastMouse := point
    }
```

```
    method onMouseDrag(point: Point) -> Done {
        if (mode == drawingMode) then {
            def aLine: Line = line.from(lastMouse) to (point) on (canvas)
            currentScribble := scribble.adding(aLine) to (currentScribble)
        } elseif (mode == movingMode) then {
            if (dragging) then {
                def diff: Point = point - lastMouse
                currentScribble.moveBy(diff.x, diff.y)
            }
        }
        lastMouse := point
    }

    method onMouseRelease(point: Point) -> Done {
        // Add code when have collection
    }

    method buttonPressed(evt:ButtonEvent) {
        if (evt.source == drawButton) then {
            mode := drawingMode
        } elseif (evt.source == moveButton) then {
            mode := movingMode
        }
    }

    startGraphics
}
```

Here is the starting code for Scribble, emptyScribble, and scribble:

```
dialect "objectdrawDialect"

type Scribble = {
    contains(pt: Point) -> Boolean

    moveBy(dx: Number, dy: Number) -> Done

}

def emptyScribble: Scribble = object {

    method contains(pt: Point) -> Boolean {
        false
    }

    method moveBy(dx: Number, dy: Number) -> Done {}
}

class scribble.adding(first': Line) to (rest': Scribble) -> Scribble {

    def first: Line = first'
    def rest: Scribble = rest'
```

```
    method contains(pt: Point) -> Boolean {
        first.contains(pt) || rest.contains(pt)
    }

    method moveBy(dx: Number, dy: Number) -> Done {
        first.moveBy(dx,dy)
        rest.moveBy(dx,dy)
    }
}
```

The start up code for the collection type, object, and class is omitted here.