Mini-Project Specification

You are to design and implement a Java application/applet called ObjectDraw. The interface of ObjectDraw should be similar to that of MiniDraw, with a drawing canvas, control panel, and toolbar/menu for selecting drawing tools. However, in ObjectDraw, objects (lines, rectangles, ovals, freehand figures) drawn on the canvas should retain their identity as distinct objects. Specifically, you should add another tool which is a “selector arrow”. Using this tool, it should be possible to “select” any one of the objects displayed on the canvas. A selected object should be highlighted in some fashion. Once a geometric object (line, rectangle, oval) has been selected, it should be possible to resize the object by clicking on one of its corners (or the corner of the bounding box for oval objects) and dragging the mouse in the direction that you wish to expand or contract the figure. It should also be possible to move a selected object by clicking on some portion of the object other than a corner and dragging the object with the mouse. The color of an object can be changed by selecting a new color on the control panel while the object is selected. When an object is selected, clicking on the eraser tool (or selecting it from the menu) should erase the object from the canvas, leaving other objects undisturbed. Clicking on an area of the control panel outside the bounds of a selected object, or clicking on the selector tool a second time, should deselect the selected object and return the application to its normal drawing function. Note that the above specification contains some ambiguities. At appropriate points in your development process, you will need to identify and resolve these ambiguities.

Mini-Project--Important Points

• You MUST employ an iterative development process.
  – Description of iterations must be handed in by Friday, Dec. 3.
    • number of iterations
    • scope and objectives of each iteration
  – At the conclusion of the project you will be expected to demonstrate the functionality of each iteration, not just the final product.
  – Your grade will be based upon how successfully you carry out your iterative development process.
Mini-Project--Important Points 
(Continued)

• You must document each step of your development process.
  – Conceptual model
  – collaboration diagram
  – discussion of design trade-offs and rationale for selected alternatives.

• You may use MiniDraw3 as a starting point.

• You do NOT need to implement Text objects.

Important Notes About Final Exam

• The final exam will be optional.
• If you elect NOT to take the final, your total points for the semester will be computed by whichever of the following methods yields the highest score:
  – Method A:
    • Midterm exams = 50% of total
      – better exam score = 30%,
      – poorer exam score = 20%
    • Homework + Mini-Project = 50% of total
      – HW assignments 1-4 = 33%
      – Mini-Project = 17%
  – Method B:
    • Midterm exams = 35% of total
      – better exam score = 20%
      – poorer exam score = 15%
    • Homework + Mini-Project = 65% of total
      – HW assignments 1-4 = 43% of total
      – Mini-Project = 22% total
Final Exam Information--Continued

• If you elect to take the final exam, your semester total will be figured as follows:
  – Exams 1-3 = 50% of total
    • best exam = 22%
    • poorest exam = 12%
    • middle exam = 16%
  – Homework + Mini-Project = 50% of total
    • HW Assignments 1-4 = 33%
    • Mini-Project = 17%

• The optional final exam will be held during the regularly scheduled final exam period.
• The final will cover material covered by the second midterm + new material introduced after the second midterm.
• You MUST notify me no later than Friday, Dec. 10, if you plan to take the final exam.

Final Exam Information--Continued

• One VERY IMPORTANT caveat regarding the optional final exam:
  – The optional final is based upon the expectation that class attendance will remain high during the remainder of the semester.
  – If class attendance falls to an unacceptable level, we will be forced to institute one or more of the following measures:
    • reinstate the mandatory final exam
    • institute unannounced quizzes during class
    • begin taking and deducting points for unexcused absences.
And One More Piece of Good News...

There will be no class on Wednesday, December 1. Use this time productively to work on the Mini-Project.

---

**MiniDraw-- Fourth Design Iteration**

- **Objective:** Add additional tools for drawing filled ovals and rectangles, and a tool for typing text.

- Note that in the third design iteration the tool classes for all of the geometric objects were very similar. All of them involve:
  - establishing an initial “end” of the object when mouse button is pressed.
  - drawing a “temporary” figure when the mouse is dragged.
  - drawing a “permanent” figure when the mouse button is released, with the cursor position at mouse release giving the second “end point” for the figure.

- The tools differ only in the specific figure drawn.
Using Subclasses to Coalesce Common Behavior of Geometric Tools

TwoEndShapeTool

LineTool | OvalTool | RecangleTool | FilledOvalTool | FilledRectangleTool

Problem: How can we move the common behavior to the superclass TwoEndShapeTool while delegating responsibility for the specific drawing operations to the appropriate subclass?
Solution: Use the GoF Strategy Pattern.

The Strategy Pattern
Use of The Strategy Pattern In MiniDraw

Tool <<interface>>

TwoEndShapeTool
TwoEndShapeTool (canvas:DrawingCanvas, shape:TwoEndShape)
MousePressed()
Mouse Dragged()
MouseReleased()
...

LineShape
draw( ... )
drawOutline( ... )

OvalShape
draw( ... )
drawOutline( ... )

RectangleShape
draw( ... )
drawOutline( ... )

TwoEndShape <<interface>>
strategy
draw(g:Graphics, x:int, y:int, width:int, height:int)
drawOutline(g:Graphics, x:int, y:int, width:int, height:int)

Adding A Text Tool to MiniDraw

• The interface for keyboard events is
  
  **KeyListener**

  KeyListener <<interface>>
  keyPressed(e:KeyEvent)
  keyReleased(e:KeyEvent)
  keyTyped(e:KeyEvent)

  KeyEvent
  ...
  getKeyChar( ):char
  ...

  KeyEvent
  ...
  getKeyChar( ):char
  ...
Keyboard Input--The Focus Issue

- Only one component can receive keyboard input at any given time.
- To receive keyboard input, a component must have the keyboard focus.
- Focus is determined by a “focus manager”.
- A component can gain keyboard focus via a requestFocus() method.
- In MiniDraw, clicking the mouse on the drawing canvas (after selecting the Text Tool) should give focus to the canvas and establish the starting point for the displayed text.

MiniDraw Fourth Iteration--Conceptual Model Extensions

```
Tool <<interface>>
   
   FreehandTool   TwoEndShapeTool
   
   TwoEndShape

   LineShape   RectangleShape   OvalShape

   KeyboardTool <<interface>>
   
   TextTool

   FilledRectangleShape   FilledOvalShape
```

Design issues:
Should we extend DrawingCanvasListener to handle KeyEvents (as shown above) or provide a separate KeyboardListener class?
Who is responsible for establishing keyboard focus?

One Possible Approach to Listening for KeyEvents:

<table>
<thead>
<tr>
<th>DrawingCanvasListener3</th>
<th>KeyListener</th>
</tr>
</thead>
<tbody>
<tr>
<td>DrawingCanvasListener4</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>keyPressed(e:KeyEvent)</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Design Issues--Resolution

• Use a separate KeyboardListener to handle KeyEvents
  – can be implemented to delegate to KeyboardTool class only.
  – This way DrawingCanvasListener doesn’t need to check the currentTool type to avoid delegating to a tool that doesn’t implement KeyListener interface.

• Have KeyboardTool establish focus in its MousePressed or MouseClicked method.
  – This way, focus is established only when currentTool is a KeyboardTool.
  – KeyboardTool has knowledge of DrawingCanvas anyway, so no new couplings are introduced.
MiniDraw Fourth Iteration--New Classes

Tool

<<interface>>

TwoEndShapeTool

TwoEndShapeTool(canvas: DrawingCanvas, shape: TwoEndShape)
mousePressed(e: MouseEvent)
mouseDragged(e: MouseEvent)
mouseReleased(e: MouseEvent)

MiniDraw Fourth Iteration Classes--Continued

TwoEndShape

<<interface>>

Draw(g: Graphics, x0: int, y0: int, x1: int, y1: int)
drawOutline (g: Graphics, x0: int, y0: int, x1: int, y1: int)

LineShape

OvalShape

RectangleShape

FilledRectangleShape

FilledOvalShape
MiniDraw Fourth Iteration Classes--Continued

KeyListener
<<interface>>

KeyListener

KeyboardListener

KeyListener(canvas: DawingCanvas)
keyPressed(e: KeyEvent)
keyReleased(e: KeyEvent)
keyTyped(e: KeyEvent)

Collaboration Diagram for
KeyPressed method

KeyPressed(e)
canvas: DrawingCanvas

: KeyboardListener

1: tool := getCurrentTool()
2: [tool != null]
   keyPressed(e)

canvas: DrawingCanvas

tool: KeyboardTool
MiniDraw Fourth Iteration Classes--Continued

Tool

<<interface>>

KeyboardTool

<<interface>>

 keyPressed(keyEvent e)
 keyreleased(keyEvent e)
 keyTyped(keyEvent e)

MiniDraw Fourth Iteration Classes--Continued

KeyboardTool

<<interface>>

TextTool

TextTool(canvas: DrawingCanvas)
mousePressed(e: MouseEvent)
keyPressed(e: KeyEvent)
{remaining methods of interface are null}
Collaboration Diagram for mousePressed method of TextTool

```
mousePressed(e)

1: requestFocus()
2: startingPos = getPoint()
3: iBGraphics = getImageBufferGraphics()
4: setFont(font)

Canvas: DrawingCanvas

iBGraphics: Graphics

text: StringBuffer

: TextTool
```

Collaboration Diagram for keyPressed method of TextTool

```
keyPressed(e)

1: nextChar = getKeyChar()
3: iBGraphics = getImageBufferGraphics()
5: drawString(str, startingPos)

Canvas: DrawingCanvas

iBGraphics: Graphics

text: StringBuffer

: TextTool
```

```
e: KeyEvent

2: append(nextChar)
4: str = toString()
```

Reaping the Benefits of Extensible Design: Adding a “Triangle Tool”

• Can draw triangle as “two end” figure:

![Diagram of triangle](image)

or

Adding a Triangle Tool--Continued

• All we need to do is create a TriangleShape class that can draw a triangle as shown on the previous page.

• Then simply add a new TwoEndShapeTool, with a TriangleShape parameter, to the ToolList.

• And we’re done.

• Total time it took me to implement it: <10 minutes.
The TriangleShape Class

```java
import java.awt.*;

public class TriangleShape implements TwoEndShape {
    public void drawOutline(Graphics g, int x0, int y0, int x1, int y1) {
        g.drawLine(x0, y0, x1, y0);
        int midX = x0 + (x1-x0)/2;
        g.drawLine(x0, y0, midX, y1);
        g.drawLine(x1, y0, midX, y1);
    }

    public void draw(Graphics g, int x0, int y0, int x1, int y1) {
        drawOutline(g, x0, y0, x1, y1);
    }
}
```

Adding the Triangle to the ToolList

```java
protected ToolList createToolList() {
    ToolList actions = new ToolList();
    
    actions.add(
        new ToolListener("Triangle",
                         getImageIcon("triangle.jpg"),
                         "Triangle drawing tool",
                         (DrawingCanvas3)canvas,
                         new TwoEndShapeTool ((DrawingCanvas3)canvas,
                                             new TriangleShape())));
    
    return actions;
}
```