9.4 Iteration 3: Refactoring

- To support the drawing of various types of shapes
  - such as lines, ovals, and rectangles
  - the design of the drawing pad must be improved to accommodate such extensions
  - to refactor the design to improve its extensibility

9.4.1 The Shapes

- The first step in refactoring the design
  - to generalize the notion of the shapes that can be drawn in the drawing pad
  - we introduce an abstract class Shape to represent all the shapes
    - not interface
    - there are fields and methods that are common to all shapes.
  -

Fig 9.10. Refactoring the scribble pad-the shapes-iteration 3 (p. 422)

Figure 9.10

Refactoring the scribble pad—the shapes.

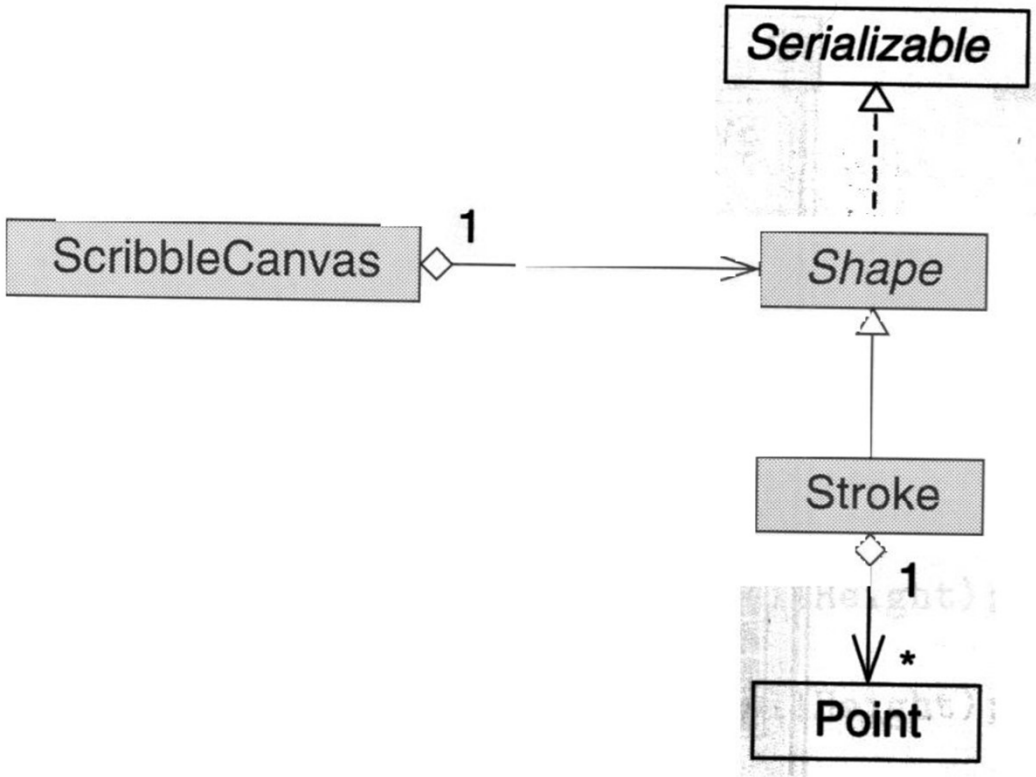
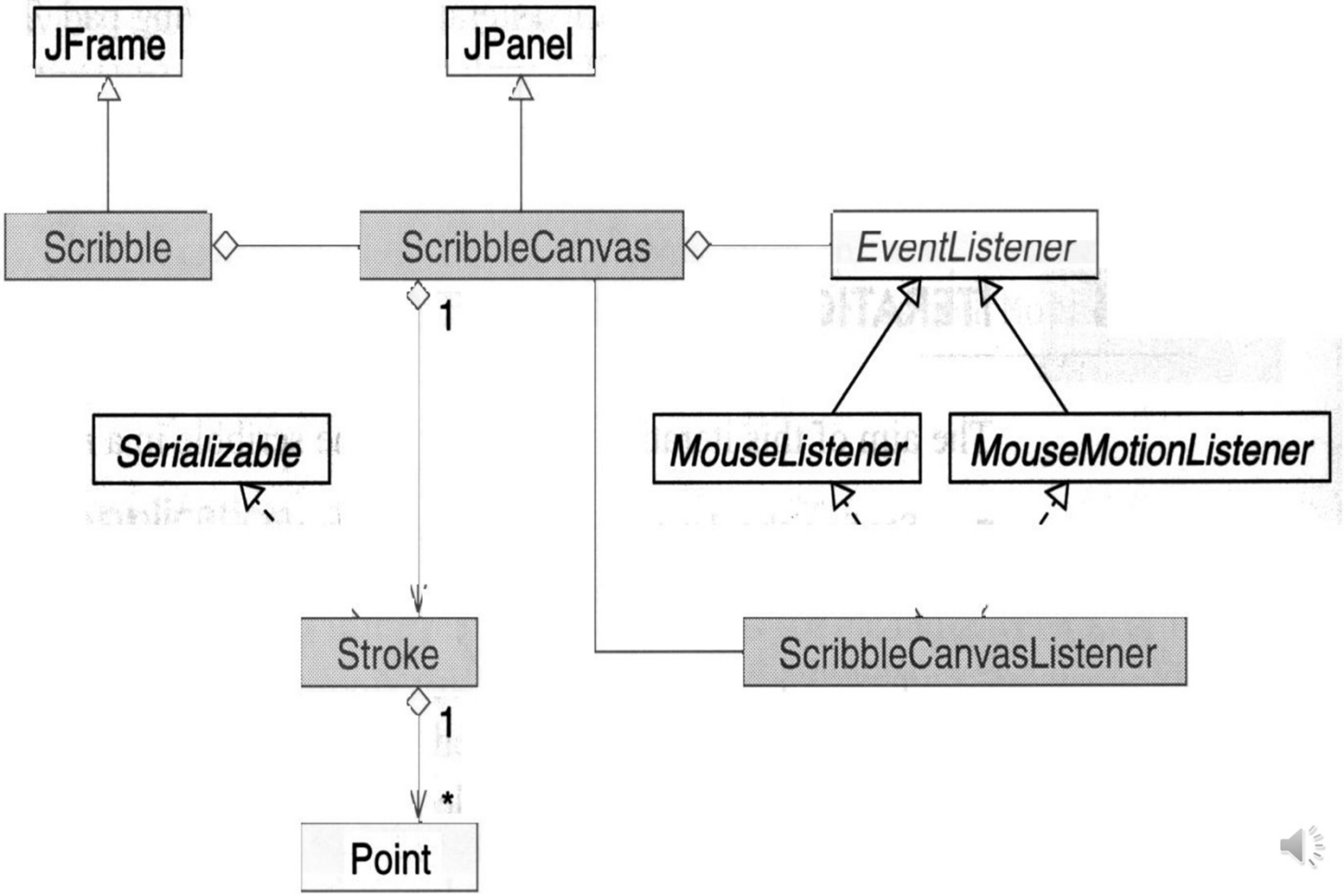


Fig 9.4 : The Design of the scribble pad – iteration 2. (p 404)

Figure 9.4

The design of the scribble pad—iteration 2.

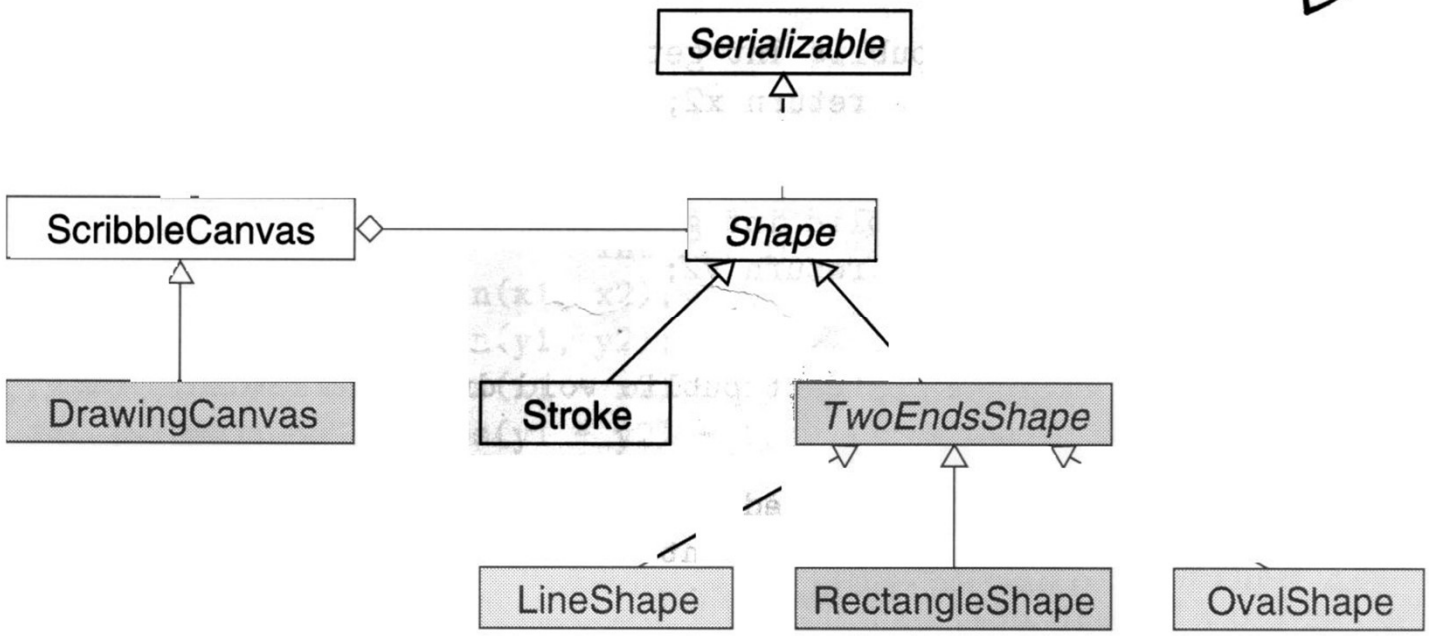


((Iteration 4))

- Figure 9.13. The design of the drawing pad – the shapes (p 433)

Figure 9.13

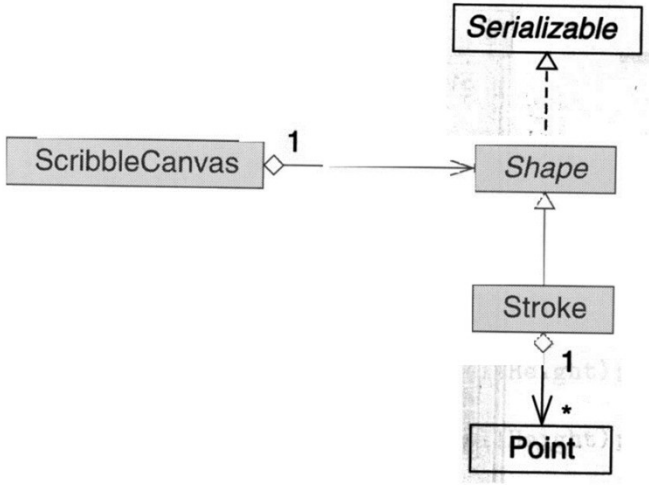
The design of the drawing pad—the shapes.



```
// Abstract Class scribble3.Shape
package scribble3;

import java.awt.*;
import java.io.Serializable;

public abstract class Shape implements Serializable {
  public Shape() {}
  public Shape(Color color) {
    this.color = color;
  }

  public void setColor(Color color) {
    this.color = color;
  }

  public Color getColor() {
    return color;
  }

  public abstract void draw(Graphics g);
  protected Color color = Color.black;
}
```

```
// Class scribble3.Stroke
package scribble3;

import java.util.*;
import java.awt.Point;
import java.awt.Color;
import java.awt.Graphics;

public class Stroke extends Shape {
  public Stroke() {}
  public Stroke(Color color) {
    super(color);
  }

  public void addPoint(Point p) {
    if (p != null) {
      points.add(p);
    }
  }

  public List getPoints() {
    return points;
  }
```

```
public void draw(Graphics g) {
   if (color != null) {
     g.setColor(color);
   }
   Point prev = null;
   Iterator iter = points.iterator();
   while (iter.hasNext()) {
     Point cur = (Point) iter.next();
     if (prev != null) {
           g.drawLine(prev.x, prev.y, cur.x, cur.y);
     }
     prev = cur;
   }
 }

 // The list of points on the stroke
 // elements are instances of java.awt.Point
 protected List points = new ArrayList();

}
 // refactoring of design concerning the shapes : Strategy Design pattern
 // Shape : abstract strategy, the Stroke class : a concrete strategy
```



9.4.2 The Tools

- The behavior of Canvas listener
  - In order to draw different shapes (such as lines, ovals, rectangles...)
  - 1) we define <u>a set of constants</u> to represent possible shapes that need to be drawn and use a field currentShape to indicate the currently selected shape (P. 424)
  - 2) the canvas listener will be extended. (p. 424)

9.4.2 The Tools

- Multiple Tools
  - an integer field . CurrrentShape
    - to indicate which tool is currently selected.

```
// P. 424
public class DrawingPad {

  // constants representing tools
  public static final int SCRIBBLE= 0;
  public static final int LINE= 1;
  public static final int RECTANGLE= 2;
  public static final int OVAL= 3;

  // the currently selected tool
  protected int currentShape = SCRIBBLE;

  // set the current tool
  public void setCurrentShape(int Shape) {
    currentShape= shape;
  }

  // get the current tool
  public int getCurrentShape() {
    return currentShape;
  }

  // other field and methods
}
```

```java
// P. 424
public class DrawingPadListener implements
                MouseListener, MouseMotionListener {

  protected ScribbleCanvas canvas;
  protected DrawingPad drawingPad;

  public ScribbleCanvasListener(ScribbleCanvas
canvas,
                DrawingPad drawingPad) {
    this.canvas = canvas;
    this.drawingPad = drawingPad
  }

  public void mousePressed(MouseEvent e) {
    Point p = e.getPoint();
    switch (drawingPad.getCurrentTool()) {
    case DrawingPad.SCRIBBLE_TOOL:
      // handle mouse pressed for the scribble tool
      break;
     case DrawingPad.LINE_TOOL:
      // handle mouse pressed for the line tool
      break;
     case DrawingPad.RECTANGLE_TOOL:
      // handle mouse pressed for the rectangle tool
      break;
     case DrawingPad.OVAL_TOOL:
      // handle mouse pressed for the oval tool
      break;
     case DrawingPad.ERASER_TOOL:
      // handle mouse pressed for the eraser tool
      break;
      }
    }
```

```java
public void mouseReleased(MouseEvent e) {
    Pont p = e.getPoint();
    switch (drawingPad.getCurrentTool()) {
    case DrawingPad.SCRIBBLE_TOOL:
      // handle mouse released for the scribble tool
      break;
     case DrawingPad.LINE_TOOL:
      // handle mouse released for the line tool
      break;
     case DrawingPad.RECTANGLE_TOOL:
      // handle mouse released for the rectangle tool
      break;
     case DrawingPad.OVAL_TOOL:
      // handle mouse released for the oval tool
      break;
     case DrawingPad.ERASER_TOOL:
      // handle mouse released for the eraser tool
      break; }
  }

  public void mouseDragged(MouseEvent e) {
    Point p = e.getPoint();
    switch (drawingPad.getCurrentTool()) {
    case DrawingPad.SCRIBBLE_TOOL:
      // handle mouse dragged for the scribble tool
      break;
     case DrawingPad.LINE_TOOL:
      // handle mouse dragged for the line tool
      break;
     case DrawingPad.RECTANGLE_TOOL:
      // handle mouse dragged for the rectangle tool
      break;
     case DrawingPad.OVAL_TOOL:
      // handle mouse dragged for the oval tool
      break;
     case DrawingPad.ERASER_TOOL:
      // handle mouse dragged for the eraser tool
      break; }
  }
  // …
}
```

9.4.2 The Tools

- Analysis of this implementation
  - cumbersome, inflexible, inelegant design
  - 1) the coupling between the DrawingPad and DrawingPadListener Class is high
    - Ex 1). The labels of the switch statements in the DrawingPadListener ... must exactly match the constant ...
    - Ex 2). Adding new tools ... require coordinating changes..
  - 2) The behavior of each tool is defined in three separate methods.. mousePressed(), mouseReleased(), and mouseDragged() ... intermixed with the behaviors of all other tools
    - changing a particular shape requires coordinated change in all three method.

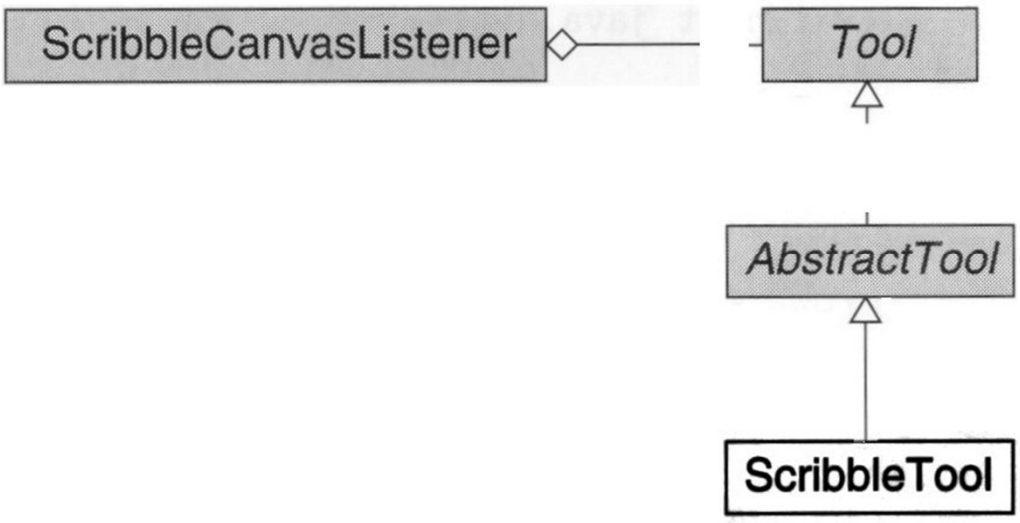
9.4.2 The Tools

- Better implementation
  - to encapsulate the behavior of each tool in a separate class (we call a tool)
    - the behavior of each tool is defined by its response to the following events: mouse mutton pressed, mouse button released, mouse dragged.
  - interface Tool : representation of the tool
  - define three concrete tools
    - ScribbleTool
    - EraserTool
    - TwoEndsTool
  -

Fig 9.11 refactoring the scribble pad – the tools (p. 425)

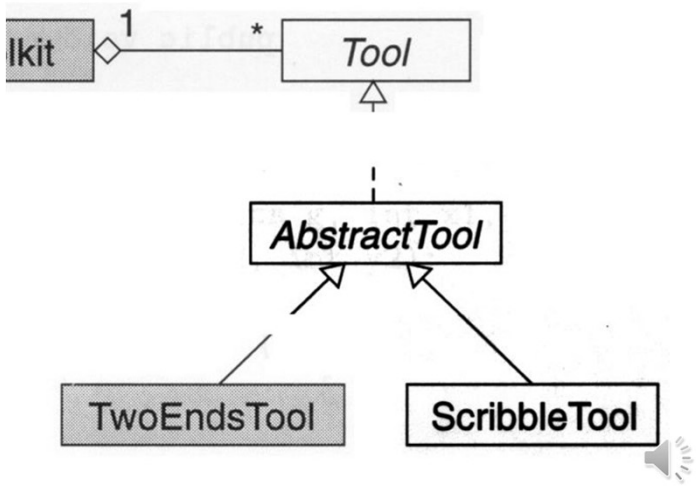
Figure 9.11

Refactoring the scribble pad—the tools.



```
// P. 426
// Interface scribble3.Tool

package scribble3;

import java.awt.*;

public interface Tool {

  public String getName();
  public void startShape(Point p);
  public void addPointToShape(Point p);
  public void endShape(Point p);

}
```

```
// P. 426
// Abstract class sribble3.AbstractTool
// implement the Tool interface and provides default implementation
// to features that are shared by all tools

package scribble3;

public abstract class AbstractTool implements Tool {

  public String getName() {
    return name;
  }

  protected AbstractTool(ScribbleCanvas canvas, String name) {
    this.canvas = canvas;
    this.name = name;
  }

  protected ScribbleCanvas canvas;
  protected String name;

}
```



```
// Class scribble3.ScribbleTool
// concrete Tool

package scribble3;

import java.awt.*;

public class ScribbleTool extends
AbstractTool {

  public ScribbleTool(ScribbleCanvas
canvas, String name) {
    super(canvas, name);
  }

  public void startShape(Point p) {
    curStroke = new
Stroke(canvas.getCurColor());
    curStroke.addPoint(p);
  }

 public void addPointToShape(Point p) {
    if (curStroke != null) {
      curStroke.addPoint(p);
      Graphics g = canvas.getGraphics();
      g.setColor(canvas.getCurColor());
      g.drawLine(canvas.x, canvas.y, p.x,
p.y);
    }
  }


  public void endShape(Point p) {
    if (curStroke != null) {
      curStroke.addPoint(p);
      canvas.addShape(curStroke);
      curStroke = null;
    }
  }

  protected Stroke curStroke = null;

}
```

```
// Class scribble3.ScribbleCanvasListener
// contains a reference to the current tool being used for drawing
// response to events of the ScribbleCanvasListener class are delegated

package scribble3;

import java.awt.*;
import java.awt.event.*;

public class ScribbleCanvasListener
    implements MouseListener, MouseMotionListener
{

  public ScribbleCanvasListener(ScribbleCanvas canvas) {
    this.canvas = canvas;
    tool = new ScribbleTool(canvas, "Scribble");
  }

  public void mousePressed(MouseEvent e) {
    Point p = e.getPoint();
    tool.startShape(p);
    canvas.mouseButtonDown = true;
    canvas.x = p.x;
    canvas.y = p.y;
  }

  public void mouseDragged(MouseEvent e) {
    Point p = e.getPoint();
    if (canvas.mouseButtonDown) {
      tool.addPointToShape(p);
      canvas.x = p.x;
      canvas.y = p.y;
    }
  }

  public void mouseReleased(MouseEvent e) {
    Point p = e.getPoint();
    tool.endShape(p);
    canvas.mouseButtonDown = false;
  }

  public void mouseClicked(MouseEvent e) {}
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mouseMoved(MouseEvent e) {}

  protected ScribbleCanvasListener(ScribbleCanvas canvas, Tool tool) {
    this.canvas = canvas;
    this.tool = tool;
  }

  protected ScribbleCanvas canvas;
  protected Tool tool;

}
```

9.4.3 Extending Components

- Two ways for Enhancements
  - 1) Modifying the original class (iteration 2, 3)
  - 2) Building a new class that extends the original class (iteration 4,5,6)
- several advantages of Extending the original class
  - 1) nondestructive :
  - 2) more suited for iterative development
  - 3) should not be modified... used by other programs

9.4.3 Extending Components

- Iteration 2, 3 : modified the original code
  - In all the subsequent iterations : use the extension techniques.
- The difference between The Scribble3.Scribble Class and the scribble2.scribble
  - in scribble3.Scribble the canvas is not created directly by using the new operator
  - rather it is created indirectly by using a factory method – makeCanvas()
  - A subclass of the Scribble can override the factory methods to create instances of the enhanced canvas.

```
// Class scribble3.Scribble
package scribble3;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;

public class Scribble extends JFrame {

  public Scribble(String title) {
    super(title);
    // calling factory method
    canvas = makeCanvas(); // factory method…
    getContentPane().setLayout(new BorderLayout());
    menuBar = createMenuBar();
    getContentPane().add(menuBar, BorderLayout.NORTH);
    getContentPane().add(canvas, BorderLayout.CENTER);
    addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
              if (exitAction != null) {
                exitAction.actionPerformed(new ActionEvent(Scribble.this, 0, null));
              }
            }
      });
  }
 // factory method : a subclass of the Scribble class can override ….
  protected ScribbleCanvas makeCanvas() {
    return new ScribbleCanvas();
  }
```

```
package scribble2;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;

public class Scribble extends JFrame {

  public Scribble() {
    setTitle("Scribble Pad");
    canvas = new ScribbleCanvas();
    getContentPane().setLayout(new BorderLayout());
    getContentPane().add(createMenuBar(), BorderLayout.NORTH);
    getContentPane().add(canvas, BorderLayout.CENTER);
    addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
              if (exitAction != null) {
                exitAction.actionPerformed(new ActionEvent(Scribble.this, 0, null));
              }
            }
      });
  }
```

9.4.3 Extending Components

- Difference between The Scribble3.ScribbleCanvas and Scribble2.ScribbleCanvas (p 406)
  - the canvas listener is also created using a factory method – makeCanvasListener()
  - the drawings are stored as a list of shapes instead of a list of strokes.

```
package scribble3;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Point;
import java.util.*;
import java.io.*;
import java.awt.event.*;
import java.util.EventListener;
import javax.swing.*;

public class ScribbleCanvas extends JPanel {

  public ScribbleCanvas() {
    // calling factory method
    listener = makeCanvasListener();
    addMouseListener((MouseListener) listener);
    addMouseMotionListener((MouseMotionListener) listener);
  }
```

```
public void newFile() {
    shapes.clear();
    repaint();
  }

  public void openFile(String filename) {
    try {
      ObjectInputStream in = new ObjectInputStream(new
FileInputStream(filename));
      shapes = (List) in.readObject();
      in.close();
      repaint();
    } catch (IOException e1) {
      System.out.println("Unable to open file: " + filename);
    } catch (ClassNotFoundException e2) {
      System.out.println(e2);
    }
  }
```

```
 public void saveFile(String filename) {
   try {
     ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(filename));
     out.writeObject(shapes);
     out.close();
     System.out.println("Save drawing to " + filename);
   } catch (IOException e) {
     System.out.println("Unable to write file: " + filename);
   }
  }
  // factory method
  protected EventListener makeCanvasListener() {
    return new ScribbleCanvasListener(this);
  }

 // The list of shapes of the drawing
  // The elements are instances of Stroke
  protected List shapes = new ArrayList();

  protected Color curColor = Color.black;
  protected EventListener listener;
  public boolean mouseButtonDown = false;
  public int x, y;
}
```

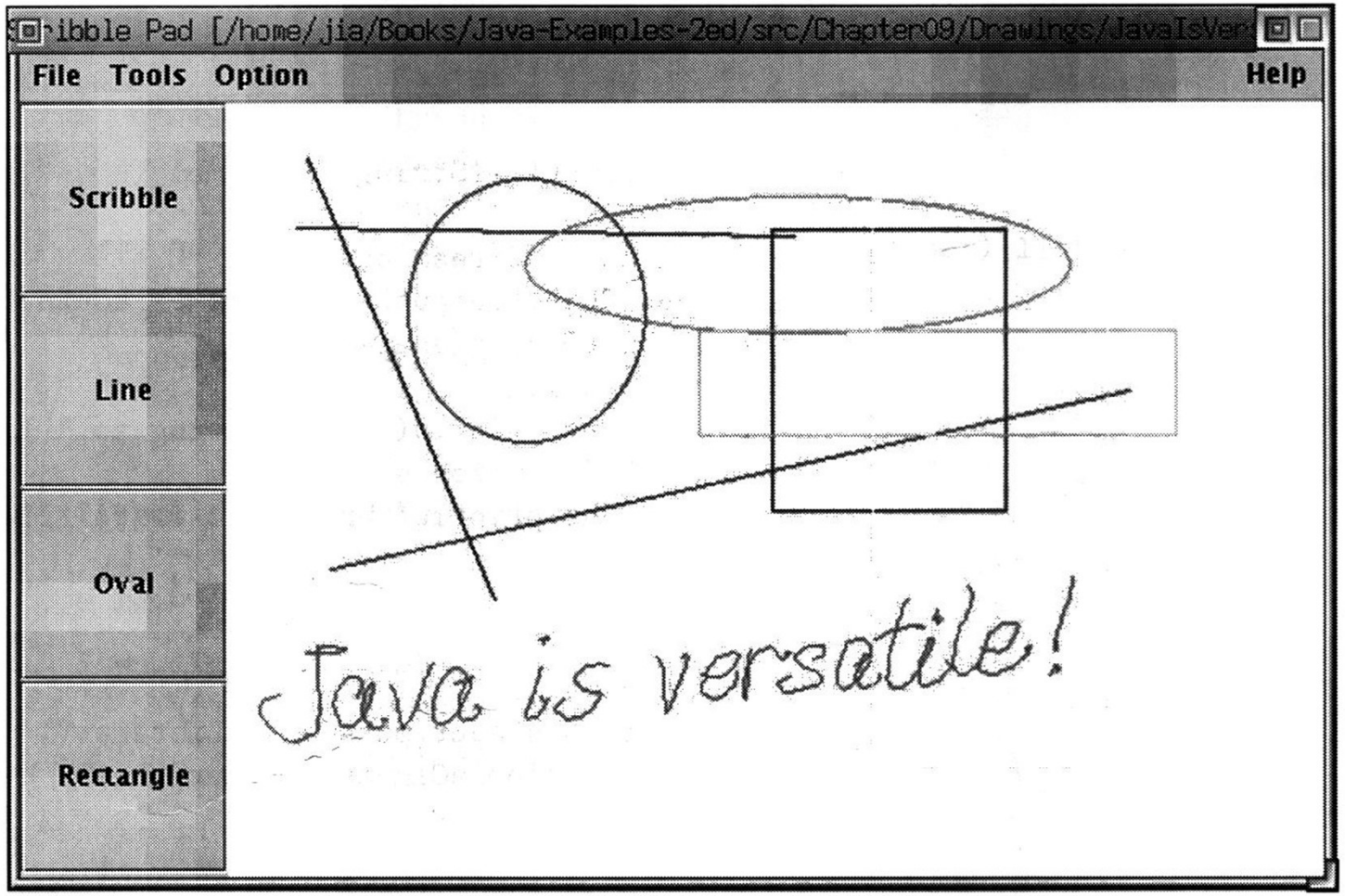
9.5 iteration 4 : adding shapes and tools

- In this iterations
  - enhance the functionality of the drawing pad
  - Fig 9.12. The drawing pad – iteration 4 (p. 432)
    - a tool bar at the left
    - scribbling tool, the line-drawing tool, the rectangle-drawing tool and the oval-drawing tool
    - adds a new menu : tools
  - the Key issues
    - Use of the State design pattern to support the different behaviors associated with different tools and to switch among different tools dynamically.
    - use of the Factory Method design pattern to allow flexibility in creating instances of different subclasses

Fig 9.12. The drawing pad – iteration 4 (p. 432)

Figure 9.12

The drawing pad—iteration 4.



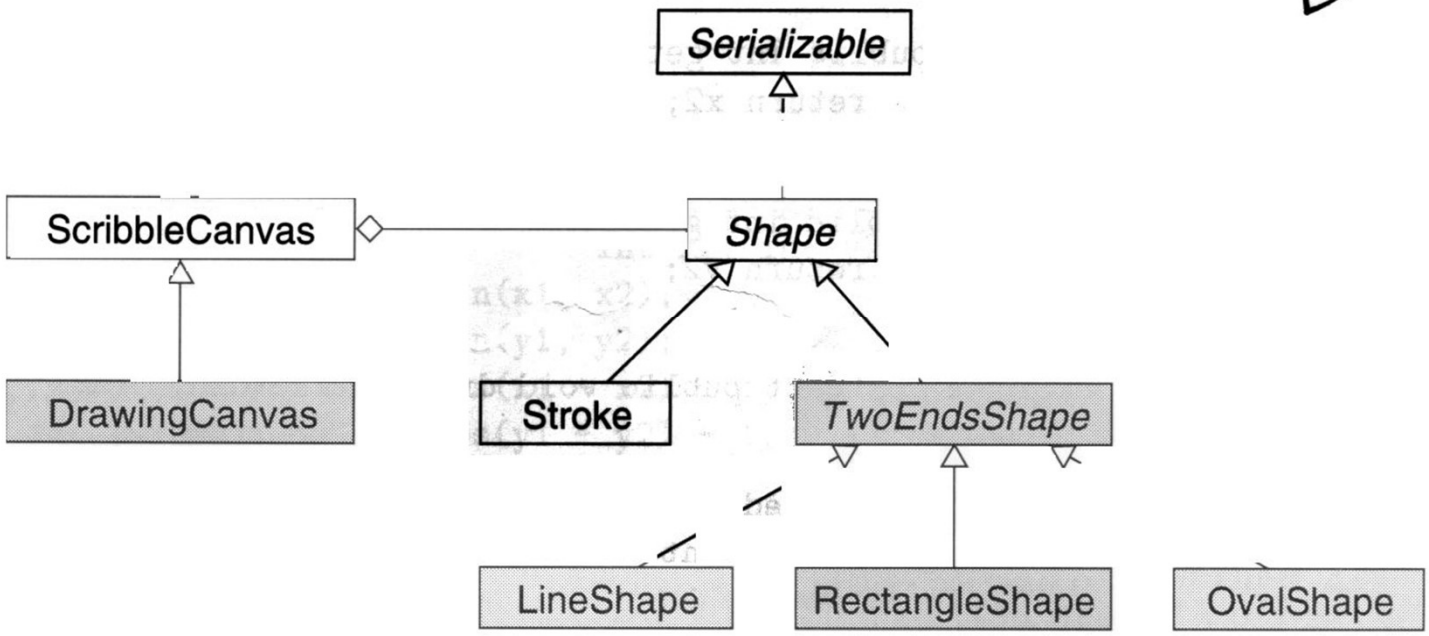
9.5.1 The Shapes

- Three types of shapes : line, oval, rectangle
  - share common characteristic
    - each of these shapes can be completely defined by two points : two end points ➔ TwoEndsShape Class
  -
  -

- Figure 9.13. The design of the drawing pad – the shapes (p 433)

Figure 9.13

The design of the drawing pad—the shapes.



- the methods of the TwoEndsShape class
  - p 433.

Methods	Description
`setEnds()`	Set both end points.
`setEnd1()`	Set the first end point.
`setEnd2()`	Set the second end point.
`getX1()`	Return the *x* coordinate of the first end point.
`getY1()`	Return the *y* coordinate of the first end point.
`getX2()`	Return the *x* coordinate of the second end point.
`getY2()`	Return the *y* coordinate of the second end point.
`drawOutline()`	Draw a temporary frame of the shape.

```
// Abstract class draw1.TwoEndsShape

package draw1;

import java.awt.Graphics;
import java.awt.Color;

public abstract class TwoEndsShape extends
scribble3.Shape implements Cloneable {

  public TwoEndsShape() {}

  public TwoEndsShape(Color color) {
    super(color);
  }

  public Object clone() throws
CloneNotSupportedException {
    return super.clone();
  }

  public void setEnds(int x1, int y1, int x2, int y2) {
    this.x1 = x1;
    this.y1 = y1;
    this.x2 = x2;
    this.y2 = y2;
  }

  public void setEnd1(int x1, int y1) {
    this.x1 = x1;
    this.y1 = y1;
  }

  public void setEnd2(int x2, int y2) {
    this.x2 = x2;
    this.y2 = y2;
  }


  public int getX1() {
    return x1;
  }

  public int getY1() {
    return y1;
  }

  public int getX2() {
    return x2;
  }

  public int getY2() {
    return y2;
  }

  abstract public void drawOutline(Graphics g, int
x1, int y1, int x2, int y2);

  protected int x1;
  protected int y1;
  protected int x2;
  protected int y2;

}
```

```
// Abstract Class sribble3.Shape
package scribble3;

import java.awt.*;
import java.io.Serializable;

public abstract class Shape implements Serializable {
  public Shape() {}
  public Shape(Color color) {
    this.color = color;
  }

  public void setColor(Color color) {
    this.color = color;
  }

  public Color getColor() {
    return color;
  }

  public abstract void draw(Graphics g);
  protected Color color = Color.black;
}
```

```
// Class draw1.LineShape

package draw1;

import java.awt.*;

public class LineShape extends TwoEndsShape {

  public void draw(Graphics g) {  //override the draw() in shape class
    if (color != null) {
      g.setColor(color);
    }
    g.drawLine(x1, y1, x2, y2);
  }

//override the drawOutline() in TwoEndsshape class
  public void drawOutline(Graphics g, int x1, int y1, int x2, int y2) {
    g.drawLine(x1, y1, x2, y2);
  }

}
```

```
// Class Draw1.OvalShape
package draw1;

import java.awt.*;
public class OvalShape extends TwoEndsShape {
  public void draw(Graphics g) {
    int x = Math.min(x1, x2);
    int y = Math.min(y1, y2);
    int w = Math.abs(x1 - x2) + 1;
    int h = Math.abs(y1 - y2) + 1;
    if (color != null) {
      g.setColor(color);
    }
    g.drawOval(x, y, w, h);
  }

  public void drawOutline(Graphics g, int x1, int y1, int x2, int y2) {
    int x = Math.min(x1, x2);
    int y = Math.min(y1, y2);
    int w = Math.abs(x1 - x2) + 1;
    int h = Math.abs(y1 - y2) + 1;
    g.drawOval(x, y, w, h);
  }
}
```

```
// Class draw1.RectangleShape

package draw1;
import java.awt.*;
public class RectangleShape extends TwoEndsShape {
  public void draw(Graphics g) {
    int x = Math.min(x1, x2);
    int y = Math.min(y1, y2);
    int w = Math.abs(x1 - x2) + 1;
    int h = Math.abs(y1 - y2) + 1;
    if (color != null) {
      g.setColor(color);
    }
    g.drawRect(x, y, w, h);
  }

  public void drawOutline(Graphics g, int x1, int y1, int x2, int y2) {
    int x = Math.min(x1, x2);
    int y = Math.min(y1, y2);
    int w = Math.abs(x1 - x2) + 1;
    int h = Math.abs(y1 - y2) + 1;
    g.drawRect(x, y, w, h);
  }
}
```

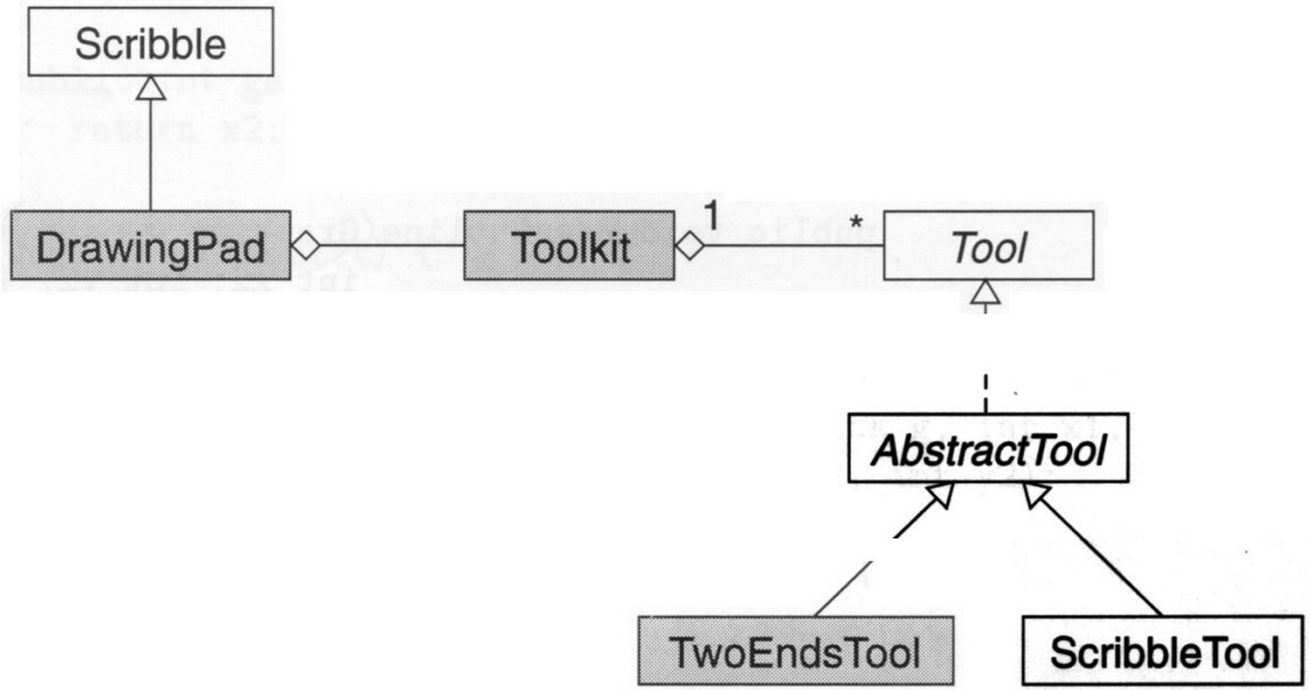
9.5.2 The Toolkit

- Toolkit Class
  - represents a set of tools supported by the drawing pad and keeps track of the currently selected tool.
  - the responses to a mouse button press and mouse dragging on the drawing canvas depend on which tool is currently selected.
  - each tool can be identified either by its name or by its position in the toolkit.

Fig 9.14. The design of the drawing pad – the tools (P. 436)

Figure 9.14

The design of the drawing pad—the tools.



the methods of the Toolkit class (P. 437)

Methods	Description
`addTool()`	Add a new tool to the toolkit.
`getToolCount()`	Return the number of tools in the toolkit.
`getTool(i)`	Return the i-th tool in the toolkit.
`findTool(name)`	Return the tool with the given name.
`setSelectedTool(i)`	Set the i-th tool to be the current tool.
`setSelectedTool(name)`	Set the tool with the given name to be the current tool.
`setSelectedTool()`	Set the specified tool to be the current tool.
`getSelectedTool()`	Return the selected tool.

```java
// Class draw1.Toolkit

package draw1;

import java.util.*;
import scribble3.Tool;

public class ToolKit {

  public ToolKit() {
  }

  /**
    Add a new tool to the tool kit.
    Return the index of the new tool.
   */
  public int addTool(Tool tool) {
    if (tool != null) {
      tools.add(tool);
      return (tools.size() - 1);
    }
    return -1;
  }

  public int getToolCount() {
    return tools.size();
  }

  public Tool getTool(int i) {
    if (i >= 0 &&
                i < tools.size()) {
      return (Tool) tools.get(i);
    }
    return null;
  }
```

```java
  public Tool findTool(String name) {
    if (name != null) {
      for (int i = 0; i < tools.size(); i++) {
                Tool tool = (Tool) tools.get(i);
                if (name.equals(tool.getName())) {
                  return tool;
                }
      }
    }
    return null;
  }
  public void setSelectedTool(int i) {
    Tool tool = getTool(i);
    if (tool != null) {
      selectedTool = tool;
    }
  }
  public Tool setSelectedTool(String name) {
    Tool tool = findTool(name);
    if (tool != null) {
      selectedTool = tool;
    }
    return tool;
  }
  public void setSelectedTool(Tool tool) {
    selectedTool = tool;
  }
  public Tool getSelectedTool() {
    return selectedTool;
  }
  protected List tools = new ArrayList(16);
  protected Tool selectedTool = null;

}
```

9.5.3 Design Pattern : State

- State design pattern
  - The refactoring concerning the toolkit
  - Allows encapsulation of the behavior of each tool in a separate class
  - and decouples the tools from the DrawingPad Class

9.5.3 Design Pattern : State

- Mouse event listener design ... the use of state design pattern.
  - able to encapsulate the behavior of each tool in a separate class...

Design Pattern ***State***

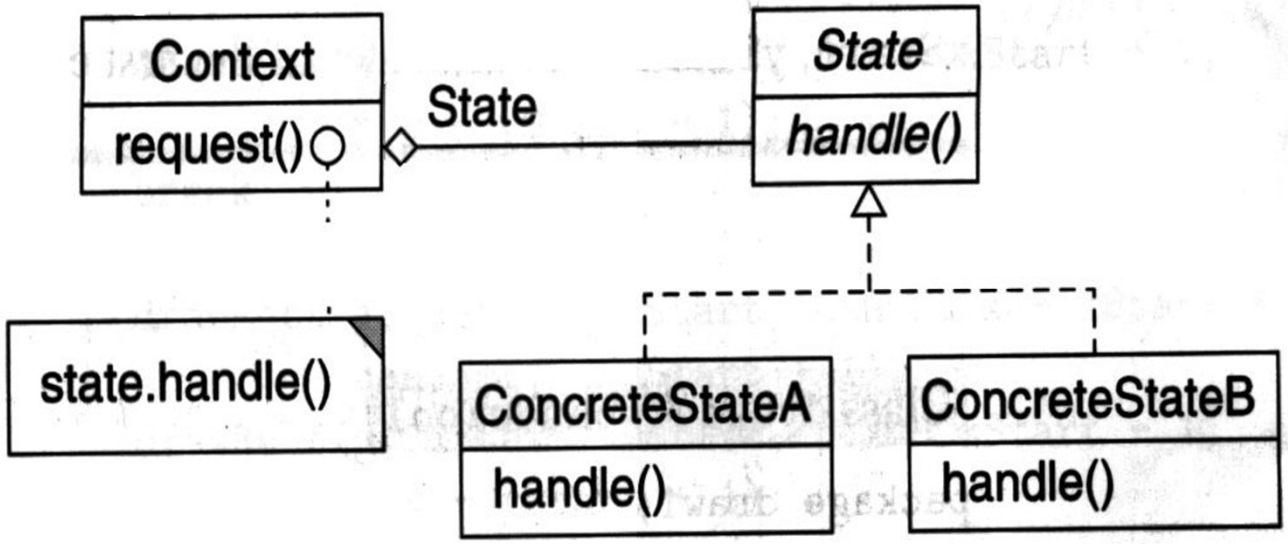
Category: Behavioral design pattern.

Intent: Allow an object to alter its behavior when its [redacted] changes.

Also Known As: Objects for [redacted]

Applicability: Use the State design pattern

- when an object's behavior depends on its state and it must change its behavior [redacted] depending on that state (e.g., selecting among several tools).
- when methods have large, multipart conditional statements that depend on the object's state (e.g., the switch statements in the `DrawingPadListener` class [p. 424]).

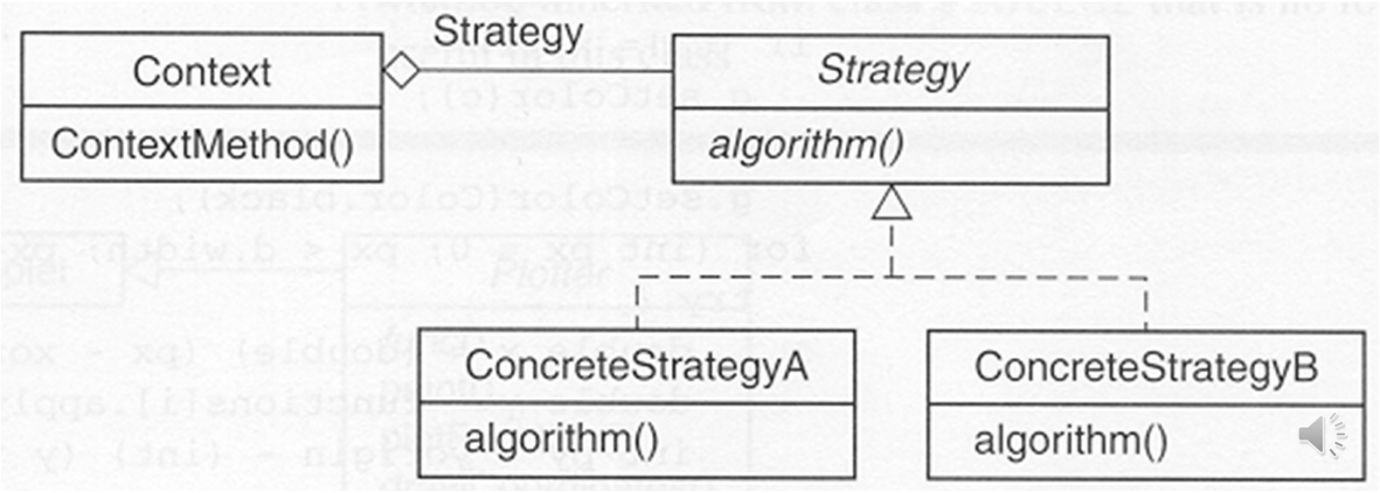


The participants of the State design pattern are as follows:

- *Context* (e.g., `DrawingPad`), which maintains an instance of a `ConcreteState` that defines the current state (e.g., the `currentTool` field in the `DrawingPad` class).
- *State* (e.g., `Tool`), which defines an interface for encapsulating the behavior associated with a particular state of the `Context`.
- *ConcreteState* (e.g., `ScribbleTool`), in which each subclass implements a behavior associated with a state of `Context`.

ActionListener of Drawing Pad

```
ActionListener toolListener = new ActionListener() {
        public void actionPerformed(ActionEvent event) {
          Object source = event.getSource();
          if (source instanceof AbstractButton) {
            AbstractButton button = (AbstractButton) source;
            Tool tool = toolkit.setSelectedTool(button.getText());
            drawingCanvas.setTool(tool);
          }
        }
    };
```



9.5.4 A Concrete Tool – TwoEndsTool

- The Two-Ends Tool
  - the two endpoints
    - similarities in drawing a line, a rectangle, a circle
    - 1) the first endpoint of the select shape
    - 2) dragging – rubber banding (temporary frames are drawn)
    - 3) release the mouse button.. The second button.
  - design a single tool (TwoEndsTool)
    - to handle all three shapes
    - each shape is identified by an integer constant : LINE, OVAL, RECT
    - fields of the TwoEndsTool (p. 440)

- fields of the TwoEndsTool (p. 440)

Field	Description
`shape`	Shape to be drawn
`xStart, yStart`	Coordinates of the first end point

```
// Package draw1.TwoEndsTool
package draw1;

import java.awt.*;
import scribble3.*;

public class TwoEndsTool extends AbstractTool {
  public static final int LINE = 0;
  public static final int OVAL = 1;
  public static final int RECT = 2;
  public TwoEndsTool(ScribbleCanvas canvas, String name, int shape) {
    super(canvas, name);
    this.shape = shape;
  }
 public void startShape(Point p) {
    canvas.mouseButtonDown = true;
    xStart = canvas.x = p.x;
    yStart = canvas.y = p.y;
    Graphics g = canvas.getGraphics();
    g.setXORMode(Color.darkGray);
    g.setColor(Color.lightGray);
    switch (shape) {
    case LINE:
      drawLine(g, xStart, yStart, xStart, yStart);
      break;
    case OVAL:
      drawOval(g, xStart, yStart, 1, 1);
      break;
    case RECT:
      drawRect(g, xStart, yStart, 1, 1);
      break;
    }
  }
```

```
public void addPointToShape(Point p) {
  if (canvas.mouseButtonDown) {
    Graphics g = canvas.getGraphics();
    g.setXORMode(Color.darkGray);
    g.setColor(Color.lightGray);
    switch (shape) {
    case LINE:
      drawLine(g, xStart, yStart, canvas.x, canvas.y);
      drawLine(g, xStart, yStart, p.x, p.y);
      break;
    case OVAL:
      drawOval(g, xStart, yStart, canvas.x - xStart + 1, canvas.y - yStart + 1);
      drawOval(g, xStart, yStart, p.x - xStart + 1, p.y - yStart + 1);
      break;
    case RECT:
      drawRect(g, xStart, yStart, canvas.x - xStart + 1, canvas.y - yStart + 1);
      drawRect(g, xStart, yStart, p.x - xStart + 1, p.y - yStart + 1);
      break;
    }
    canvas.x = p.x;
    canvas.y = p.y;
  }
}
```

```
public void endShape(Point p) {
  canvas.mouseButtonDown = false;
  TwoEndsShape newShape = null;
  switch (shape) {
  case LINE:
    newShape = new LineShape();
    break;
  case OVAL:
    newShape = new OvalShape();
    break;
  case RECT:
    newShape = new RectangleShape();
  }
  if (newShape != null) {
    newShape.setColor(canvas.getCurColor());
    newShape.setEnds(xStart, yStart, p.x, p.y);
    canvas.addShape(newShape);
  }
  Graphics g = canvas.getGraphics();
  g.setPaintMode();
  canvas.repaint();
}
protected int shape = LINE;
protected int xStart, yStart;
```

```
// helper methods
 public static void drawLine(Graphics g, int x1, int y1, int x2, int y2) {
   g.drawLine(x1, y1, x2, y2);
 }
 public static void drawRect(Graphics g, int x, int y, int w, int h) {
   if (w < 0) {
     x = x + w;
     w = -w;
   }
   if (h < 0) {
     y = y + h;
     h = -h;
   }
   g.drawRect(x, y, w, h);
 }
```

```
public static void drawOval(Graphics g, int x, int y, int w, int h) {
    if (w < 0) {
      x = x + w;
      w = -w;
    }
    if (h < 0) {
      y = y + h;
      h = -h;
    }
    g.drawOval(x, y, w, h);
  }
}
```

9.5.4 A Concrete Tool – TwoEndsTool

- Rubber banding
  - exclusive OR mode of the graphic context
  - mousePressed()
    - prepare for rubber banding
  - mouseDragged()
    - performs the rubber banding
  - mouseReleased()
    - records the second end point of the shape
    - sets the mode of the graphics context to the default paint mode.

9.5.5 Extending Components

- Fig 9.15. The overall design of the drawing pad – iteration 4 (p. 443)
  - factory methods are used in the extended classes to create <u>instances of the extended classes</u>