Object –Oriented Software Development Using Java

Chap 9. Design Case Study: A Drawing Pad

한동대학교 전자전산학부

김 기석 교수

- peterkim@handong.edu
- http://www.handong.edu

9.1 Planning

- Chapter Overview
  - case study in developing GUI, Using JFC
  - iterative development process in successive increment
  - two design patterns : Factory Method and State

- Iterative Development Process
  - Each iteration involves a development cycle
  - including conceptualization, analysis and modeling, design and implementation.
  - each iteration results in a completely functional intermediate product
  - the use of design patterns : crucial roles

9.1 Planning

- Goal : Drawing Tool
  - using lightweight components in the Swing package
  - Scribbling and drawing various shapes
  - saving the drawings to files and loading the drawings from file
  - typing from the keyboard
  - Choosing colors and fonts
- Six successive iterations
  - 1) creates a simple scribble pad, which consists of only a canvas for scribbling
  - 2) adding support for saving and loading drawings, a menu bar, various dialog
  - 3) Refactoring to support various tools for drawing various shapes
  - 4) Adding Tools for drawing lines, rectangles, and ovals
  - 5) Refactoring and adding tools for drawing filled rectangles and ovals
  - 6) Adding a tool for typing text

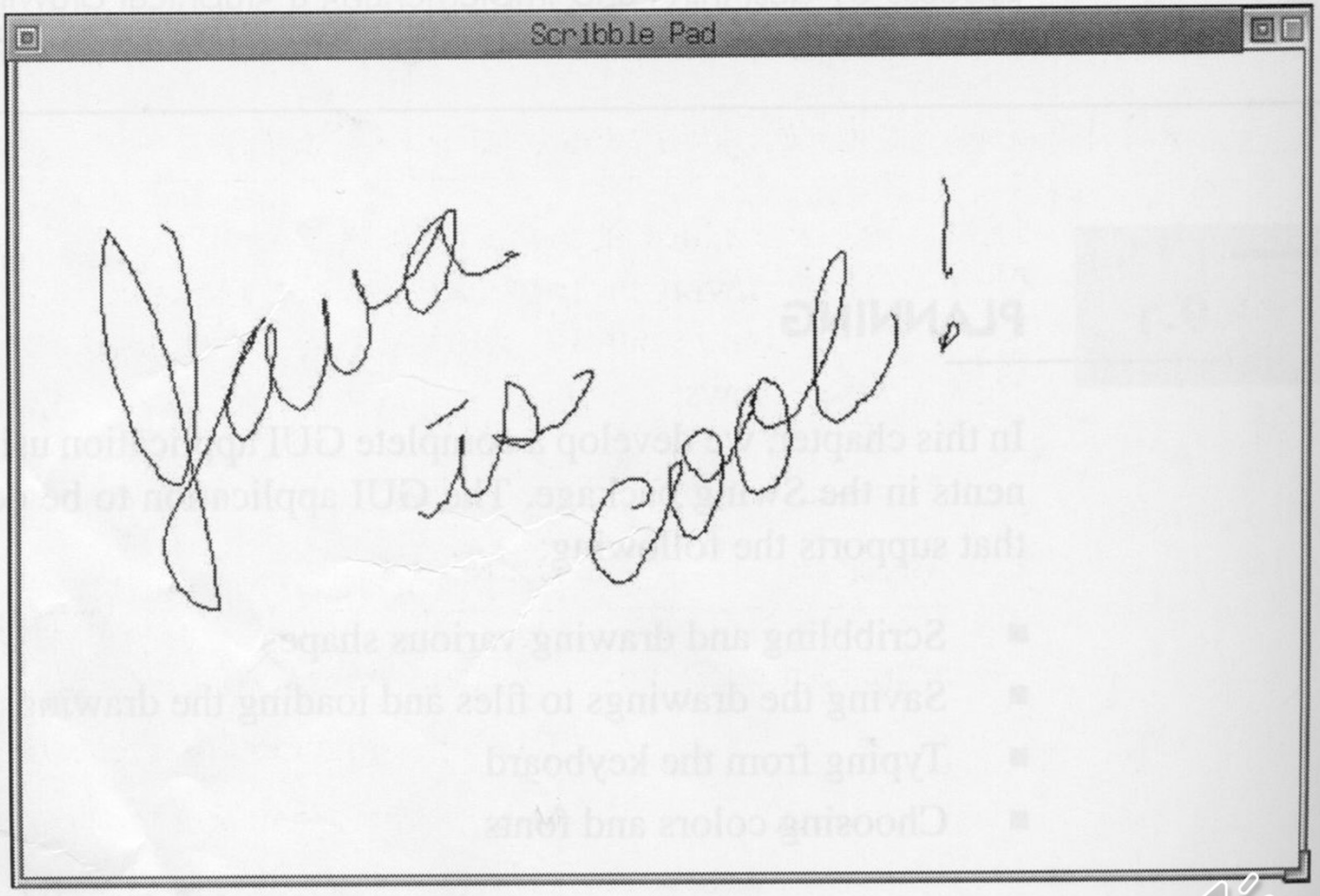
9.2 Iteration 1-A Simple Scribbling Pad

- Simple Scribble Pad, Scribble Pad
  - only supporting scribbling
  - the application consists of a frame that contains the canvas for scribbling only.

Figure 9.1 The simple scribble pad (P 398) : Demo

Figure 9.1

The simple scribble pad.

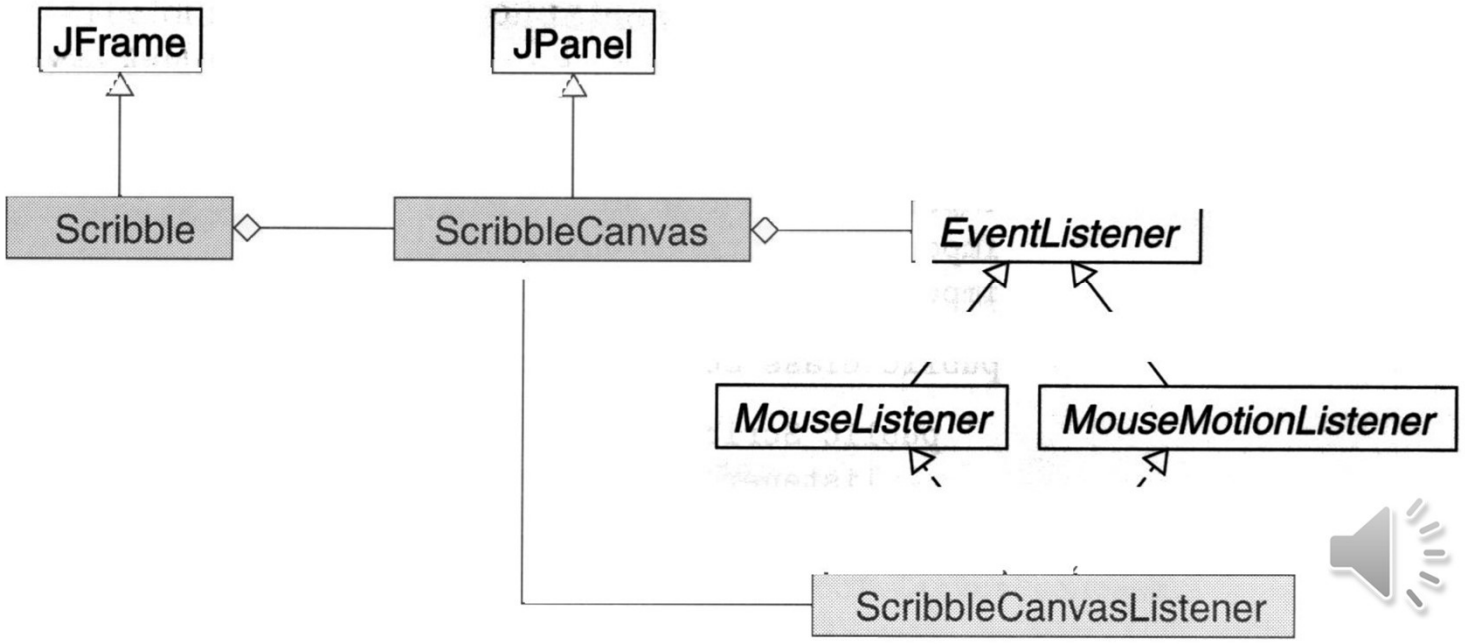


9.2 Iteration 1–A Simple Scribbling Pad

- Figure 9.2 The structure of the simple scribble pad (P 399)
  - Shaded Classes : to be developed in the current iteration
  - The rest : classes and interfaces either in the Java 2 Class Library or developed in previous iteration

Figure 9.2

The design of the scribble pad—iteration 1.



- Class to be developed in scribble1 (P 399)

Class	Description
`Scribble`	The main application
`ScribbleCanvas`	The canvas for scribble
`ScribbleCanvasListener`	The event listener that listens to mouse events for scribbling

9.2.1 The Scribbling Canvas and its Listener

- ScribbleCanvas Class
  - The GUI component on which all the drawing takes place.
  - extends the javax.swing.swing.Jpanel
  - Fields of the ScribblingCanvas Class

Fields	Description
`mouseButtonDown`	Whether one of the mouse buttons is being pressed down
`x, y`	The current mouse position in the canvas
`listener`	Mouse event listener of the canvas

```
// Class scribble1.ScribbleCanvas

package scribble1;

import java.awt.*;
import java.awt.event.*;
import java.util.EventListener;
import javax.swing.*;

public class ScribbleCanvas extends JPanel {

  public ScribbleCanvas() {
    listener = new ScribbleCanvasListener(this);
    addMouseListener((MouseListener) listener);
    addMouseMotionListener((MouseMotionListener) listener);
  }

  protected EventListener listener;
  protected boolean mouseButtonDown = false; // whether one of the
mouse button is being pressed down
  protected int x, y;

}
```

9.2.1 The Scribbling Canvas and its Listener

- ScribbleCanvasListener Class
  - handle the mouse events that occur in the canvas
  - The desired behaviors
    - A stroke begins when any mouse button is pressed
    - the stroke continues when the mouse is dragged
    - the stroke finished when the mouse button is released
  - Two event Listener
    - 1) : button press, release
    - 2) : movement of mouse (mouse move or drag)

- Relevant methods of the ScribbleCanvasListener class

Methods	Description
`mousePressed()`	Notified when a mouse button is pressed
`mouseReleased()`	Notified when a mouse button is released
`mouseDragged()`	Notified while the mouse is being dragged

```
// Class scribble1.ScribbleCanvasListener

package scribble1;

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

public class ScribbleCanvasListener
    implements MouseListener, MouseMotionListener {

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

 // mousePressed : beginning of a stroke
  public void mousePressed(MouseEvent e) {
    Point p = e.getPoint();
                  // getPoint() : the current mouse position is obtained
    canvas.mouseButtonDown = true;
    canvas.x = p.x;
    canvas.y = p.y;
  }
```

```
public void mouseReleased(MouseEvent e) {
    canvas.mouseButtonDown = false;
  }

  // mouseDragged () : handles the continuation of a stroke
  public void mouseDragged(MouseEvent e) {
    Point p = e.getPoint();
    if (canvas.mouseButtonDown) {
      canvas.getGraphics().drawLine(canvas.x, canvas.y, p.x, p.y);
            // It draws a line from the previous mouse position to the current
            // and mouse position
      canvas.x = p.x;
      canvas.y = p.y;
    }
  }

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

  protected ScribbleCanvas canvas;

}
```

Class JComponent

- getGraphics
- public Graphics getGraphics()
  - Returns this component's graphics context, which lets you draw on a component. Use this method to get a Graphics object and then invoke operations on that object to draw on the component.
  - Overrides: getGraphics in class Component
  - Returns: this components graphics context
  - See Also: Component.paint(java.awt.Graphics)

Class Graphics

- drawLine
- public abstract void drawLine(int x1, int y1, int x2, int y2)
  - Draws a line, using the current color, between the points (x1, y1) and (x2, y2) in this graphics context's coordinate system.
  - Parameters: x1 – the first point's *x* coordinate. y1 – the first point's *y* coordinate. x2 – the second point's *x* coordinate. y2 – the second point's *y* coordinate.

9.2.2. The Application

- The Scribble Class
  - main application class of the initial version of the scribble pad
  - extends the javax.swing.Jframe
  - A frame is created in the main()
  - an instance of the ScribbleCanvas is placed at the center of the frame

```
// Class scribble1.Scribble

package scribble1;

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

public class Scribble extends JFrame {

  public Scribble() {
    setTitle("Scribble Pad");
    canvas = new ScribbleCanvas();
    canvas.setBackground(Color.white);
    getContentPane().setLayout(new BorderLayout());
    getContentPane().add(canvas, BorderLayout.CENTER);
    addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
              System.exit(0);
            }
      });
  }
```

```
public static void main(String[] args) {
   int width = 600;
   int height = 400;

   // A Frame is created
   JFrame frame = new Scribble();

   frame.setSize(width, height);
   Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

   // place the application frame at the center of the screen
   frame.setLocation(screenSize.width/2 - width/2,
                     screenSize.height/2 - height/2);
   frame.show();
 }

 protected ScribbleCanvas canvas;

}
```

Class Toolkit

- public abstract class Toolkit
  - extends Object
  - This class is the abstract superclass of all actual implementations of the Abstract Window Toolkit. Subclasses of Toolkit are used to bind the various components to particular native toolkit implementations.
  - Most applications should not call any of the methods in this class directly. The methods defined by Toolkit are the "glue" that joins the platform-independent classes in the java.awt package with their counterparts in java.awt.peer. Some methods defined by Toolkit query the native operating system directly.

Methods of Toolkit

- static Toolkit getDefaultToolkit()
  Gets the default toolkit.
- abstract Dimension getScreenSize()
  Gets the size of the screen.

Class JFrame

- getContentPane
- public Container getContentPane()
  - Returns the contentPane object for this frame.
  - Specified by:
    - getContentPane in interface RootPaneContainer
  - Returns:
    - the contentPane property
  - See Also:
    - setContentPane(java.awt.Container), RootPaneContainer.getContentPane()

9.3 ITERATION 2 : Menus, Options, and Files

- The aim of this iteration
  - Storing the drawings internally so that they can be redrawn
  - Saving the drawings into files and loading drawings from files
  - Building a menu bar
  - Using the file dialogs
  - creating a dialog box for selecting colors

Fig 9.3 : The Scribble Pad – iteration 2 (p 403)

Figure 9.3

The scribble pad—iteration 2.

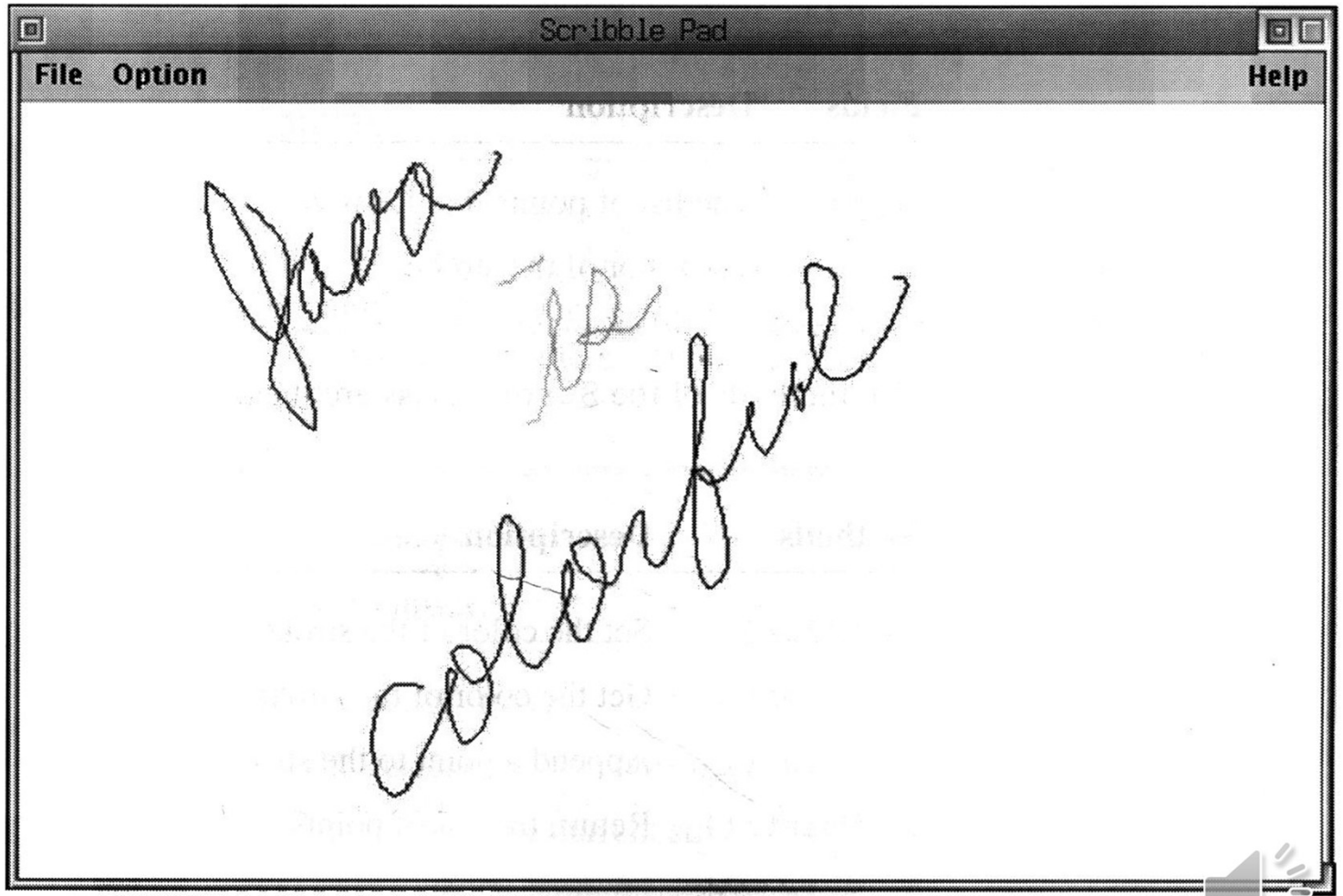
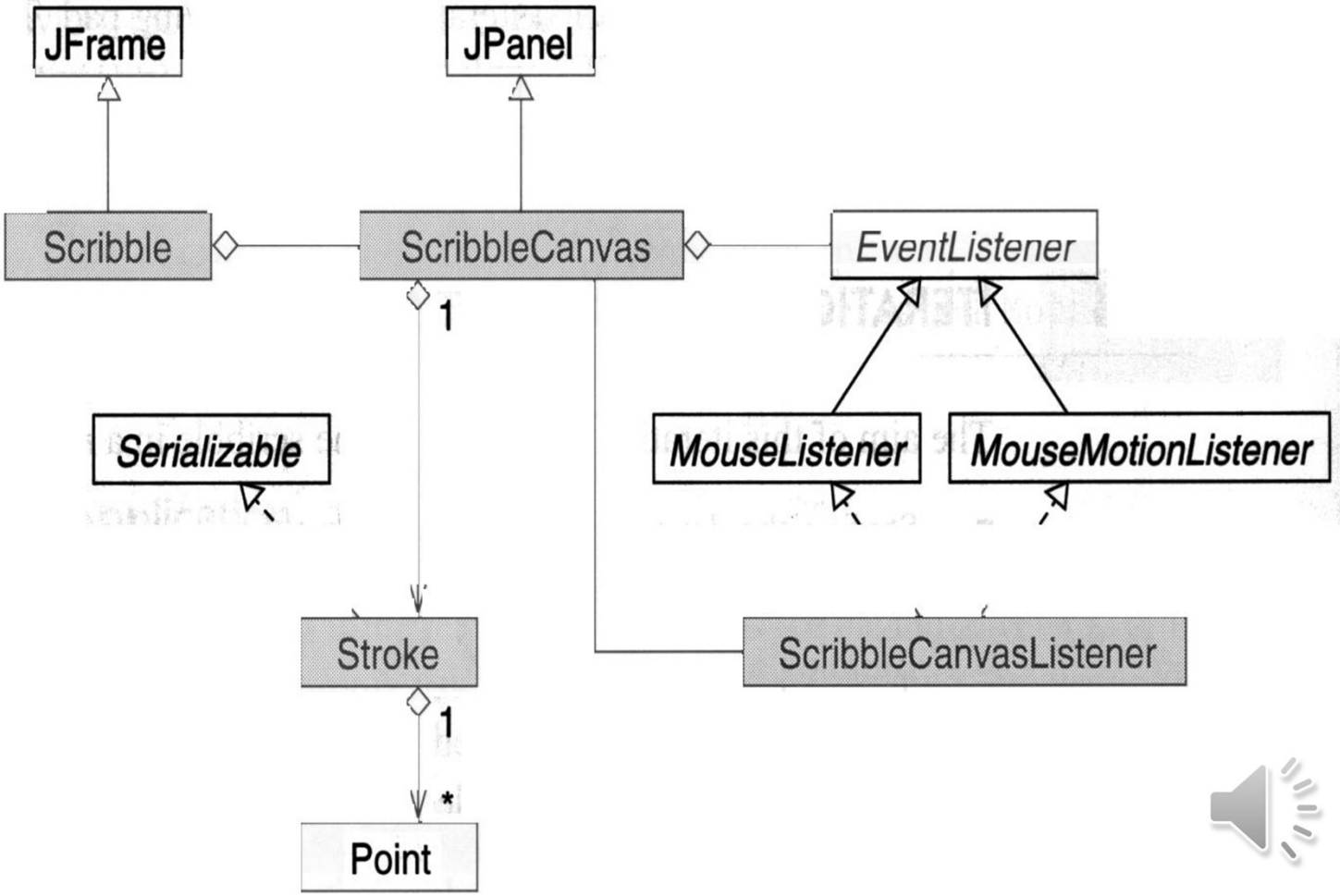


Fig 9.4 : The Design of the scribble pad – iteration 2. (p 404)

Figure 9.4

The design of the scribble pad—iteration 2.



9.3.1 Strokes

- Previous version
  - the drawings are not stored internally
  - the drawings disappear
- The Drawings are stored in the following structure
  - Each drawing consists of a list of strokes
  - Each stroke consists of a list of points and the color of the stroke.

9.3.1 Strokes

- Scribble2.Stroke class
  - represent a single stroke in a drawing
  - to save the drawings in files.. We make Stroke class serializable so that we can use the serialization mechanism to save and load drawings.

- the fields of the Stroke Class (p. 404)

Fields	Description
`points`	The list of points that form the stroke
`color`	The color of the stroke

- the methods of the Stroke Class (p. 404)

Methods	Description
`setColor()`	Set the color of the stroke.
`getColor()`	Get the color of the stroke.
`addPoint()`	Append a point to the stroke.
`getPoints()`	Return the list of points.

```
// Class scribble2.Stroke
package scribble2;

import java.util.*;
import java.io.Serializable;
import java.awt.Point;
import java.awt.Color;

public class Stroke implements Serializable {

  public Stroke() { }

  public Stroke(Color color) {
    this.color = color;
  }

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

  public Color getColor() {
    return color;
  }
```

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

  public List getPoints() {
    return points;
  }

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

  protected Color color = Color.black;

}
```

9.3.2 The Scribble Canvas

- The ScribbleCanvas class
  - stores the drawing as a list of strokes
  - repaints the drawings onto the canvas whenever necessary
  - fields for ScribbleCanvas (p.405)

Fields	Description
`strokes`	The list of strokes
`curStroke`	The current stroke, while the stroke is being drawn
`curColor`	The color of the current stroke

- the methods of the scribble2.ScribbleCanvas (p 406)

Methods	Description
`setCurColor()`	Set the current color.
`getCurColor()`	Get the current color.
`startStroke()`	Invoked by the listener to start a new stroke. A new current stroke is created.
`addPointToStroke()`	Invoked by the listener to append a point to the current stroke. A new point is appended to the current stroke.
`endStroke()`	Invoked by the listener to end a new stroke. The current stroke is added to the list of strokes.
`paint()`	Invoked by the Java run time whenever the canvas needs to be painted. The drawing is repainted onto the canvas by retracing the strokes stored internally.
`newFile()`	Create a new drawing and a new file.
`saveFile()`	Save the current drawing to a file using object serialization.
`openFile()`	Load a drawing from a file by deserializing the object stored in the file and repaint the canvas.

```
// Scribble2.ScribbleCanvas Class
package scribble2;

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() {
    listener = new
ScribbleCanvasListener(this);
    addMouseListener((MouseListener)
listener);

addMouseMotionListener((MouseMotionLis
tener) listener);
  }
  public void setCurColor(Color curColor) {
    this.curColor = curColor;
  }
 public Color getCurColor() {
    return curColor;
  }

// A new current Stroke is created
  public void startStroke(Point p) {
    curStroke = new Stroke(curColor);
    curStroke.addPoint(p);
  }

// A new point is appended to the current
stroke
public void addPointToStroke(Point p) {
    if (curStroke != null) {
      curStroke.addPoint(p);
    }
  }

// the current stroke is added to the list of
strokes
 public void endStroke(Point p) {
    if (curStroke != null) {
      curStroke.addPoint(p);
      strokes.add(curStroke);
      curStroke = null;
    }
  }
```

```
// the drawing is repainted onto the canvas by retracing the strokes stored internally
public void paint(Graphics g) {
    Dimension dim = getSize();
    g.setColor(Color.white);
    g.fillRect(0, 0, dim.width, dim.height);
    g.setColor(Color.black);
    if (strokes != null) {
      Iterator iter1 = strokes.iterator();
      while (iter1.hasNext()) {
            Stroke stroke = (Stroke) iter1.next();
            if (stroke != null) {
              g.setColor(stroke.getColor());
              Point prev = null;
              List points = stroke.getPoints();
              Iterator iter2 = points.iterator();
              while (iter2.hasNext()) {
                Point cur = (Point) iter2.next();
                if (prev != null) {
                  g.drawLine(prev.x, prev.y, cur.x, cur.y);
                }
                prev = cur;
              }
            }
    }
  }
}
```

```
// Create a new drawing and a new file.
  public void newFile() {
    strokes.clear();
    repaint();
  }

// Load a drawing from a file by
deserializing the object stored in the file
and repaint the canvas.
  public void openFile(String filename) {
    try {
      ObjectInputStream in = new
ObjectInputStream(new
FileInputStream(filename));
      strokes = (List) in.readObject();
      in.close();
      repaint();
    } catch (IOException e1) {
      System.out.println("Unable to open file:
" + filename);
    } catch (ClassNotFoundException e2) {
      System.out.println(e2);
    }
  }
```

```
 // Save the current drawing to a file using
object serialization
 public void saveFile(String filename) {
    try {
      ObjectOutputStream out = new
ObjectOutputStream(new
FileOutputStream(filename));
      out.writeObject(strokes);
      out.close();
      System.out.println("Save drawing to " +
filename);
    } catch (IOException e) {
      System.out.println("Unable to write file: " +
filename);
    }
  }

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

  // curStroke : the current Stroke, while the
stroke is being drawn
  protected Stroke curStroke = null;
  protected Color curColor = Color.black;

  protected EventListener listener;
  protected boolean mouseButtonDown = false;
  protected int x, y;

}
```

9.3.3 The Canvas Listener

- scribble2.ScribbleCanvasListener Class
  - difference
    - in addition to drawing the points of a stroke onto the canvas, the points must also be stored in strokes.

```
// p 401과 비교 요망
package scribble2;

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

public class ScribbleCanvasListener
    implements MouseListener, MouseMotionListener {

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

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

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

```
public void mouseDragged(MouseEvent e) {
    Point p = e.getPoint();
    if (canvas.mouseButtonDown) {
      canvas.addPointToStroke(p);
      Graphics g = canvas.getGraphics();
      g.setColor(canvas.getCurColor());
      g.drawLine(canvas.x, canvas.y, p.x, p.y);
      canvas.x = p.x;
      canvas.y = p.y;
    }
  }

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

  protected ScribbleCanvas canvas;

}
```

9.3.4 The Application

- Main application Class : scribble2.scribble class
  - adding a menu bar
  - a number of nested class
    - listeners of the menu items
  -

- the methods of the scribble2.Scribble class (p. 410)

Methods	Description
`createMenuBar()`	Create the menu bar.
`newFile()`	Start a new drawing and a new file.
`openFile()`	Load a drawing from the specified file.
`saveFile()`	Save the current drawing to the current file.
`saveFileAs()`	Save the current drawing to the specified file.

- the nested classes of the scribble2.scribble class (p. 410)

Nested Classes	Description
`NewFileListener`	Action for the "New" menu item
`OpenFileListener`	Action for the "Open" menu item
`SaveFileListener`	Action for the "Save" menu item
`SaveAsFileListener`	Action for the "Save As" menu item
`ExitListener`	Action for the "Exit" menu item
`ColorListener`	Action for the "Color" menu item
`AboutListener`	Action for the "About" menu item

```
// Class scribble2.Scribble
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));
              }
            }
      });
  }
```

```
//…. Nested claSSES….


public static void main(String[] args) {
    int width = 600;
    int height = 400;
    JFrame frame = new Scribble();
    frame.setSize(width, height);
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    frame.setLocation(screenSize.width/2 - width/2,
                      screenSize.height/2 - height/2);
    frame.show();
  }

  protected String currentFilename = null;
  protected ScribbleCanvas canvas;
  protected ActionListener exitAction;
  protected JFileChooser chooser = new JFileChooser(".");

}
```

```
// the menu bar is created
// Method of scribble2.scribble class :
createMenuBar()

protected JMenuBar createMenuBar() {
    JMenuBar menuBar = new JMenuBar();
    JMenu menu;
    JMenuItem mi;

    // File menu
    menu = new JMenu("File");
    menuBar.add(menu);

    mi = new JMenuItem("New");
    menu.add(mi);
    mi.addActionListener(new NewFileListener());

    mi = new JMenuItem("Open");
    menu.add(mi);
    mi.addActionListener(new OpenFileListener());

    mi = new JMenuItem("Save");
    menu.add(mi);
    mi.addActionListener(new SaveFileListener());

    mi = new JMenuItem("Save As");
    menu.add(mi);
    mi.addActionListener(new
SaveAsFileListener());

    menu.add(new JSeparator());

exitAction = new ExitListener();
    mi = new JMenuItem("Exit");
    menu.add(mi);
    mi.addActionListener(exitAction);

    // option menu
    menu = new JMenu("Option");
    menuBar.add(menu);

    mi = new JMenuItem("Color");
    menu.add(mi);
    mi.addActionListener(new ColorListener());

    // horizontal space
    menuBar.add(Box.createHorizontalGlue());

    // Help menu
    menu = new JMenu("Help");
    menuBar.add(menu);

    mi = new JMenuItem("About");
    menu.add(mi);
    mi.addActionListener(new AboutListener());

    return menuBar;
  }
```

public class Box extends Container implements Accessible

- A lightweight container that uses a BoxLayout object as its layout manager. Box provides several class methods that are useful for containers using BoxLayout -- even non-Box containers.
- The Box class can create several kinds of invisible components that affect layout: glue, struts, and rigid areas. If all the components your Box contains have a fixed size, you might want to use a glue component (returned by createGlue) to control the components' positions. If you need a fixed amount of space between two components, try using a strut (createHorizontalStrut or createVerticalStrut). If you need an invisible component that always takes up the same amount of space, get it by invoking createRigidArea.
- static ComponentcreateHorizontalGlue()
  Creates a horizontal glue component.

```
// Nested Class of scribble2.Scribble class : AboutListener
// using javax.swing.JOptionPane (Figure 9.5)

class AboutListener implements ActionListener {

    public void actionPerformed(ActionEvent e) {
      JOptionPane.showMessageDialog(null,
            "DrawingPad version 1.0\nCopyright (c) Xiaoping Jia 2002",
"About",
            JOptionPane.INFORMATION_MESSAGE);
    }

  }

// P 413. Figure 9.5
```

Figure 9.5

The "About" dialog of the scribble pad.

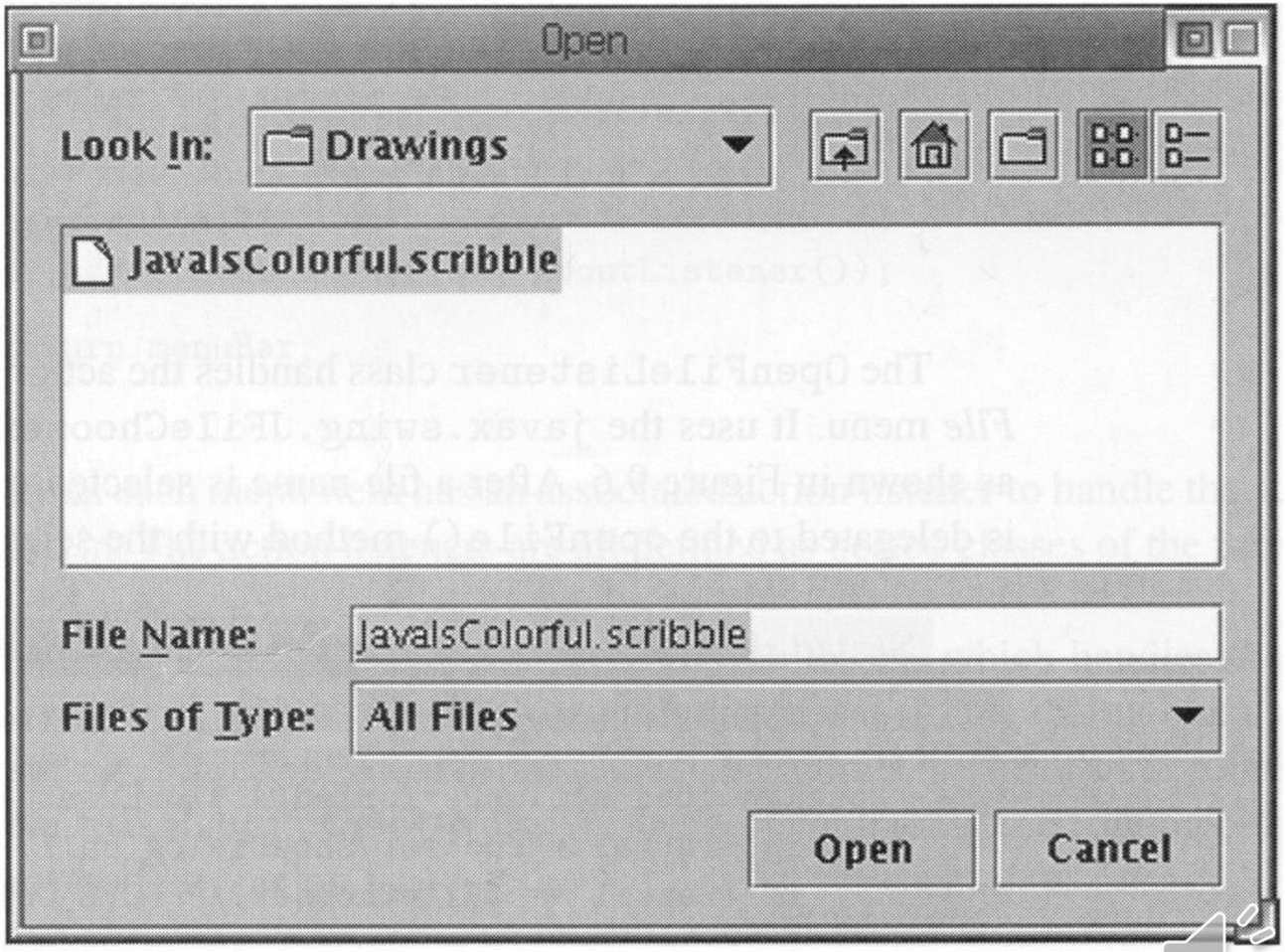
Class JOptionPane

- JOptionPane makes it easy to pop up a standard dialog box that prompts users for a value or informs them of something. While the class may appear complex because of the large number of methods, almost all uses of this class are one-line calls to one of the static showXxxDialog methods shown below:
  - showConfirmDialog : Asks a confirming question, like yes/no/cancel.
  - showInputDialog : Prompt for some input.
  - showMessageDialog : Tell the user about something that has happened.
  - showOptionDialog : The Grand Unification of the above three.

```
// Nested Class of scribble2.Scribble class : NewFIleListener
class NewFileListener implements ActionListener {

  public void actionPerformed(ActionEvent e) {
    newFile();
  }
}

// Nested class of scribble2.Scribble class : OpenFileListener
class OpenFileListener implements ActionListener {
  public void actionPerformed(ActionEvent e) {
  //protected JFileChooser chooser = new JFileChooser(".");
    int retval = chooser.showDialog(null, "Open"); // Fig 9.6
    if (retval == JFileChooser.APPROVE_OPTION) {
          File theFile = chooser.getSelectedFile();
          if (theFile != null) {
            if (theFile.isFile()) {
              String filename = chooser.getSelectedFile().getAbsolutePath();
              openFile(filename);
            }
          }
    }
  }
}
```

Figure 9.6

The file dialog of the scribble pad.



Class JFileChooser

- JFileChooser provides a simple mechanism for the user to chooser a file.

Constructor of JFileChooser

- public JFileChooser(File currentDirectory)
  - Creates a JFileChooser using the given File as the path. Passing in a null file causes the file chooser to point to the users's home directory.
  - Parameters: directory – a File object specifying the path to a file or directory

Method of JFileChooser

- showDialog
  - public int showDialog(Component parent, String approveButtonText)
  - Pops a custom file chooser dialog with a custom ApproveButton. e.g. filechooser.showDialog(parentWindow, "Run Application"); would pop up a filechooser with a "Run Application" button (instead of the normal "Save" or "Open" button). Alternatively, the following code will do the same thing: JFileChooser chooser = new JFileChooser(null); chooser.setApproveButtonText("Run Application"); chooser.showDialog(this, null); PENDING(jeff) - the following method should be added to the api: showDialog(Component parent);
  - Parameters: approveButtonText - the text of the ApproveButton
  - Returns: the return state of the filechooser on popdown: CANCEL_OPTION, APPROVE_OPTION

Method of JFileChooser

- File getSelectedFile()
  Returns the selected file.
- File[] getSelectedFiles()
  Returns a list of selected files if the filechooser is set to allow multi-selection.

Method of Class File

- String getAbsolutePath()
  Returns the absolute pathname string of this abstract pathname.

```
 // Nested Class of scribble2.Scribble Class : SaveFileListener
class SaveFileListener implements ActionListener {

    public void actionPerformed(ActionEvent e) {
      saveFile();
    }

  }
// Nested Class of scribble2.Scribble Class : SaveAsFileListener
class SaveAsFileListener implements ActionListener {

    public void actionPerformed(ActionEvent e) {
      int retval = chooser.showDialog(null, "Save As"); // Fig 9.6
      if (retval == JFileChooser.APPROVE_OPTION) {
            File theFile = chooser.getSelectedFile();
            if (theFile != null) {
              if (!theFile.isDirectory()) {
                String filename = chooser.getSelectedFile().getAbsolutePath();
                saveFileAs(filename);
              }
            }
      }
    }

  }
```

```
// Nested Class of scribble2.Scribble Class : ExitListener
 class ExitListener implements ActionListener {

    public void actionPerformed(ActionEvent e) {   // Fig 9.7
      int result = JOptionPane.showConfirmDialog(null,
                                  "Do you want to exit Scribble Pad?",
                                  "Exit Scribble Pad?",
                                   JOptionPane.YES_NO_OPTION);
      if (result == JOptionPane.YES_OPTION) {
            saveFile();
            System.exit(0);
      }
    }

  }
```

Figure 9.7

The "Exit" dialog of the scribble pad.

Methods of JOptionPane

- static intshowConfirmDialog(Component parentCompo nent, Object message) : Brings up a modal dialog with the options Yes, No and Cancel; with the title, "Select an Option".
- static intshowConfirmDialog(Component parentCompon ent, Object message, String title, int optionType) : Brings up a modal dialog where the number of choices is determined by the optionType parameter.
- static intshowConfirmDialog(Component parentCompon ent, Object message, String title, int optionType, int messageType) : Brings up a modal dialog where the number of choices is determined by the optionType parameter, where the messageType parameter determines the icon to display.

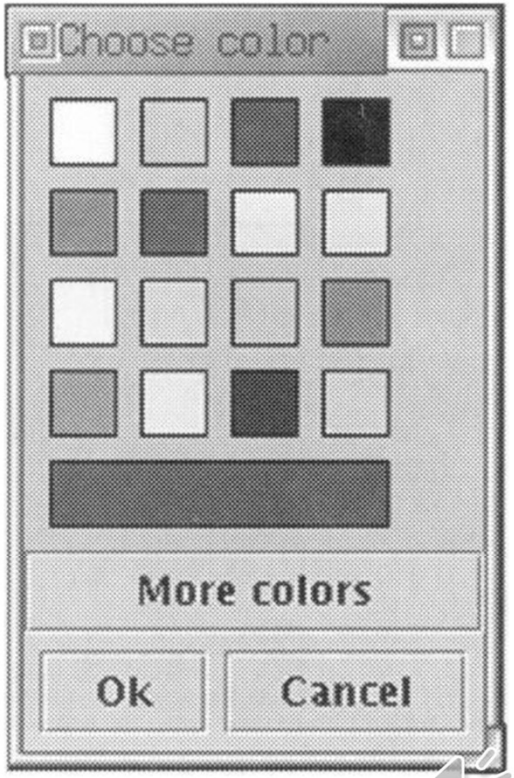
```
protected void newFile() {
    currentFilename = null;
    canvas.newFile();
    setTitle("Scribble Pad");
  }

  protected void openFile(String filename) {
    currentFilename = filename;
    canvas.openFile(filename);
    setTitle("Scribble Pad [" + currentFilename + "]");
  }

  protected void saveFile() {
    if (currentFilename == null) {
      currentFilename = "Untitled";
    }
    canvas.saveFile(currentFilename);
    setTitle("Scribble Pad [" + currentFilename + "]");
  }

  protected void saveFileAs(String filename) {
    currentFilename = filename;
    canvas.saveFile(filename);
    setTitle("Scribble Pad [" + currentFilename + "]");
  }
```

9.3.5 Choosing Colors

- The nested ColorListener class
  - handles the action of the Color menu item in the Option menu
  - displays a custom dialog box (P. 416. Fig 9.8)

Figure 9.8

The color dialog of the scribble pad.



```
class ColorListener implements ActionListener {

   public void actionPerformed(ActionEvent e) {
     Color result = dialog.showDialog();
     if (result != null) {
           canvas.setCurColor(result);
     }
   }

   protected ColorDialog dialog =
     new ColorDialog(Scribble.this, "Choose color", canvas.getCurColor());

 }

// Class ColorDiaglog : p. 417 – 420.
// Fig 9.9 The Swing Color
// Javax.swing.ColorChooser

protected JColorChooser chooser = new JColorChooser();  // p 418
```

Figure 9.9

The Swing color chooser.