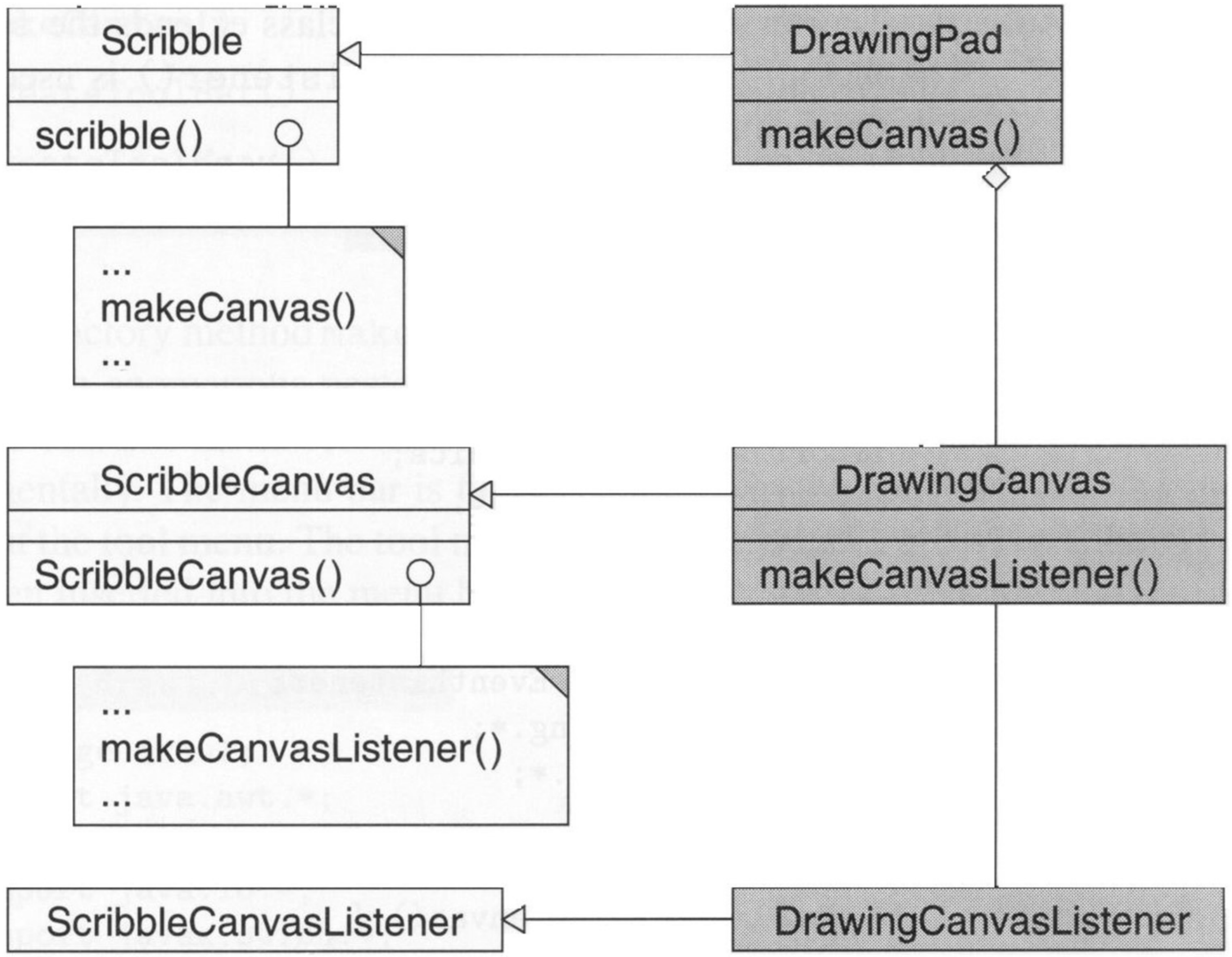
9.5.5 Extending Components

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

Figure 9.15

The overall design of the drawing pad—iteration 4.



```
// scribble3.ScribbleCanvasListener class(p. 427) support only one tool,
the scribble tool
// the draw1.DrawingCanvasListener class : the tool can be set to any tool

package draw1;

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

public class DrawingCanvasListener extends ScribbleCanvasListener {

  public DrawingCanvasListener(DrawingCanvas canvas) {
    super(canvas, null);
  }

  public Tool getTool() {
    return tool;
  }

  public void setTool(Tool tool) {
    this.tool = tool;
  }

}
```

```
// the factory method makeCanvasListener() is used to create an instance
of the enhanced listener

// Class draw1.DrawingCanvas

package draw1;

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.*;
import scribble3.*;
```

```
public class DrawingCanvas extends ScribbleCanvas {
                                   // ScribbleCanvas : p. 430.(Slide #19)
  public DrawingCanvas() {
  }

  public void setTool(Tool tool) {
    drawingCanvasListener.setTool(tool);
  }

  public Tool getTool() {
    return drawingCanvasListener.getTool();
  }

  // factory method
  protected EventListener makeCanvasListener() {
    return (drawingCanvasListener = new DrawingCanvasListener(this));
  }

  protected DrawingCanvasListener drawingCanvasListener;

}
```

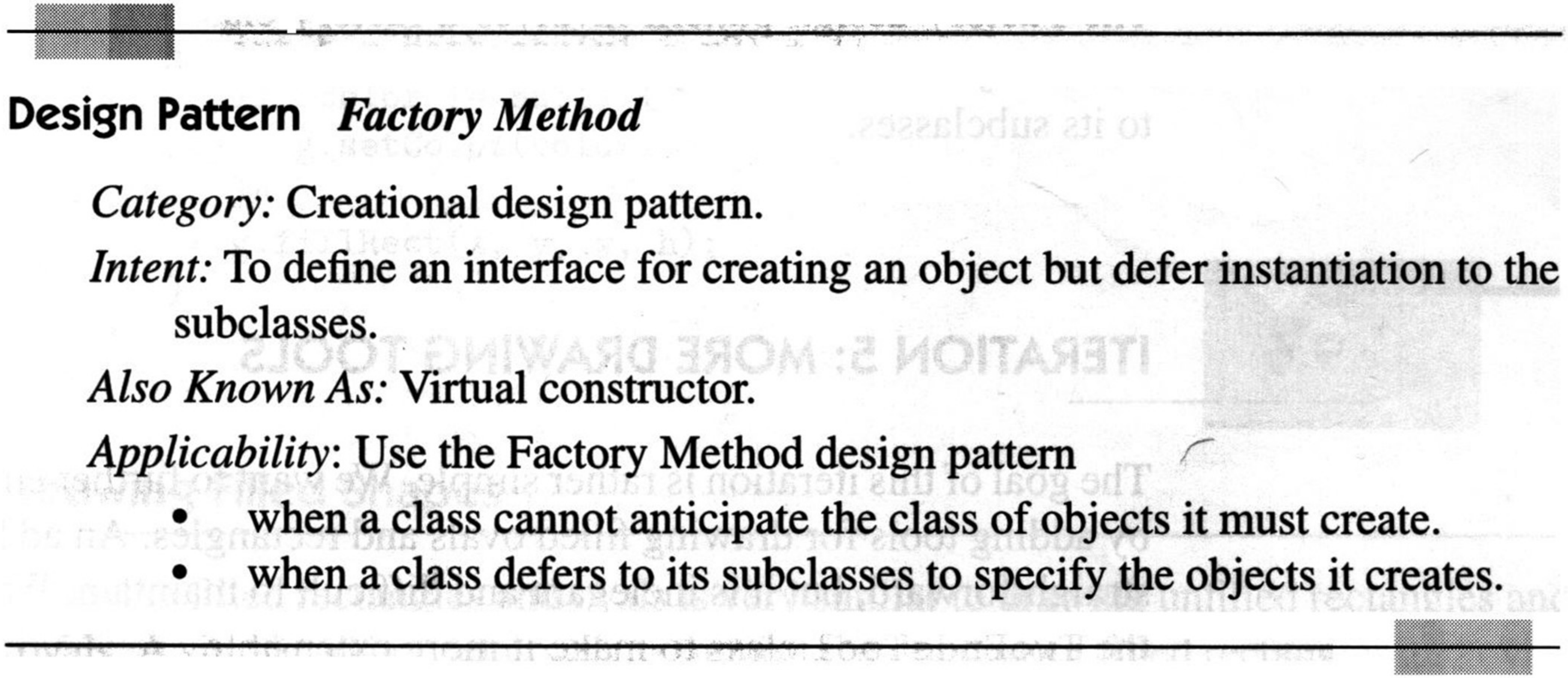
```
// Class draw1.DrawingPad
// Main Application Class… manages the toolkit, creates the tool bar and
the tool menu
package draw1;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
import scribble3.*;

public class DrawingPad extends Scribble {
  public DrawingPad(String title) {
    super(title);
    initTools();
    // an anonymous nested class to select the current tool
    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);
              }
            }
     };
     // create the tool bar
    JComponent toolbar = createToolBar(toolListener);
    getContentPane().add(toolbar, BorderLayout.WEST);

    // create the tool menu
    JMenu menu = createToolMenu(toolListener);

    // insert the tool menu into the menu bar
    menuBar.add(menu, 1); // insert at index position 1
  }
   public Tool getSelectedTool() {
    return toolkit.getSelectedTool();
  }
 // create the drawing tools and initialize the toolkit
  protected void initTools() {
    toolkit = new ToolKit();
    toolkit.addTool(new ScribbleTool(canvas, "Scribble"));
    toolkit.addTool(new TwoEndsTool(canvas, "Line", TwoEndsTool.LINE));
    toolkit.addTool(new TwoEndsTool(canvas, "Oval", TwoEndsTool.OVAL));
    toolkit.addTool(new TwoEndsTool(canvas, "Rectangle",
TwoEndsTool.RECT));
    drawingCanvas.setTool(toolkit.getTool(0));
  }
```

```
// factory method (Scribble : p. 429,Slide#19)
 protected ScribbleCanvas makeCanvas() {
   return (drawingCanvas = new DrawingCanvas());
 }

 protected JComponent createToolBar(ActionListener toolListener) {
   JPanel toolbar = new JPanel(new GridLayout(0, 1));
   int n = toolkit.getToolCount();
   for (int i = 0; i < n; i++) {

     // create a button for each tool
     Tool tool = toolkit.getTool(i);
     if (tool != null) {
          JButton button = new JButton(tool.getName());
          button.addActionListener(toolListener);
          toolbar.add(button);
     }
   }
   return toolbar;
 }
```

```
protected JMenu createToolMenu(ActionListener toolListener) {
   JMenu menu = new JMenu("Tools");

  // create a menu item for each tool
   int n = toolkit.getToolCount();
   for (int i = 0; i < n; i++) {
     Tool tool = toolkit.getTool(i);
     if (tool != null) {
          JMenuItem menuitem = new JMenuItem(tool.getName());
          menuitem.addActionListener(toolListener);
          menu.add(menuitem);
     }
   }
   return menu;
 }
```

```
protected ToolKit toolkit;
  protected DrawingCanvas drawingCanvas;

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

}
```

9.5.6 Design Pattern : Factory Method



Design Pattern ***Factory Method***

Category: Creational design pattern.

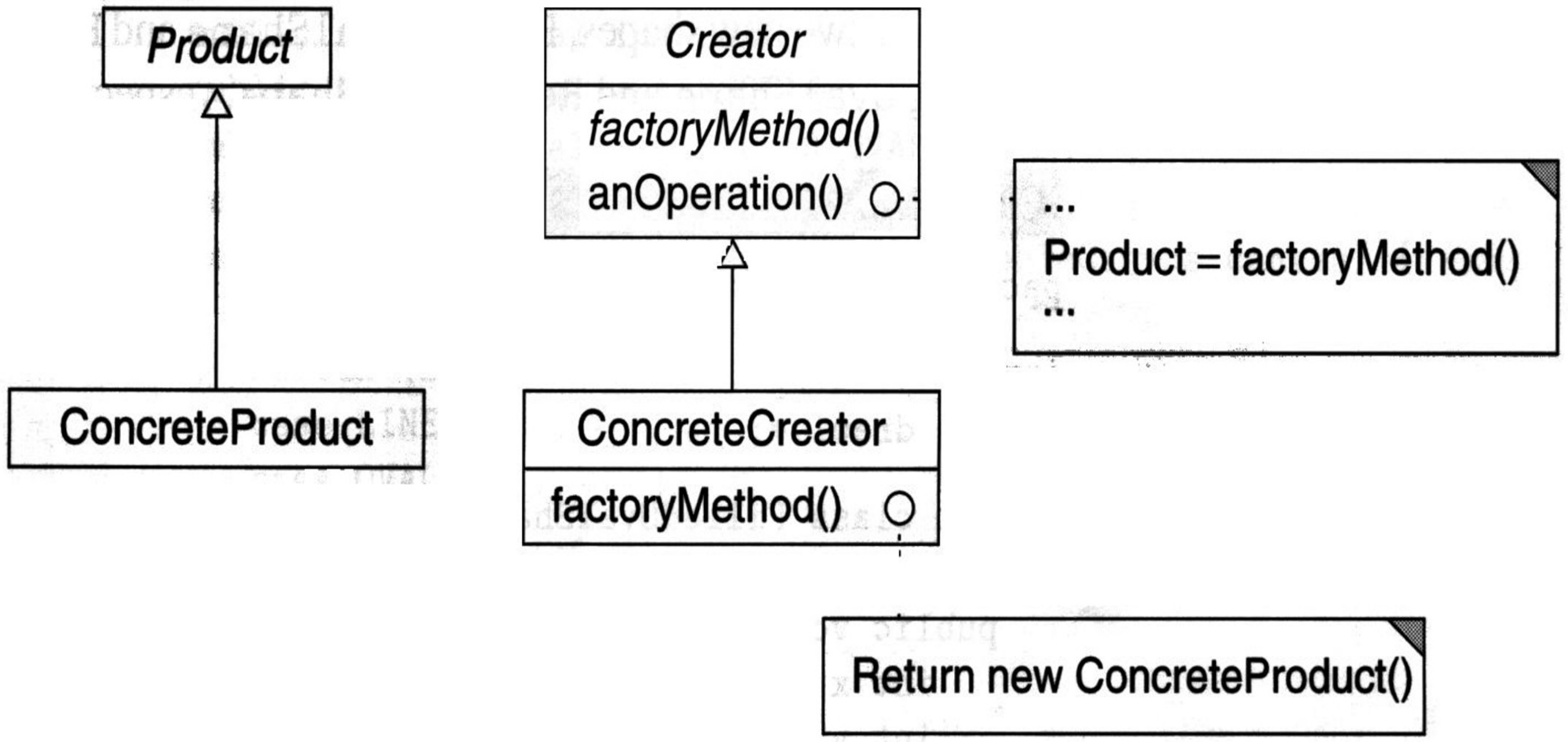
Intent: To define an interface for creating an object but defer instantiation to the subclasses.

Also Known As: Virtual constructor.

Applicability: Use the Factory Method design pattern

- when a class cannot anticipate the class of objects it must create.
- when a class defers to its subclasses to specify the objects it creates.

9.5.6 Design Pattern : Factory Method



- *Product* (e.g., `EventListener`), which defines the interface of the objects to be created.
- *ConcreteProduct* (e.g., `ScribbleCanvasListener` and `ToolListener`), which implements the *Product* interface, and may provide default implementation.

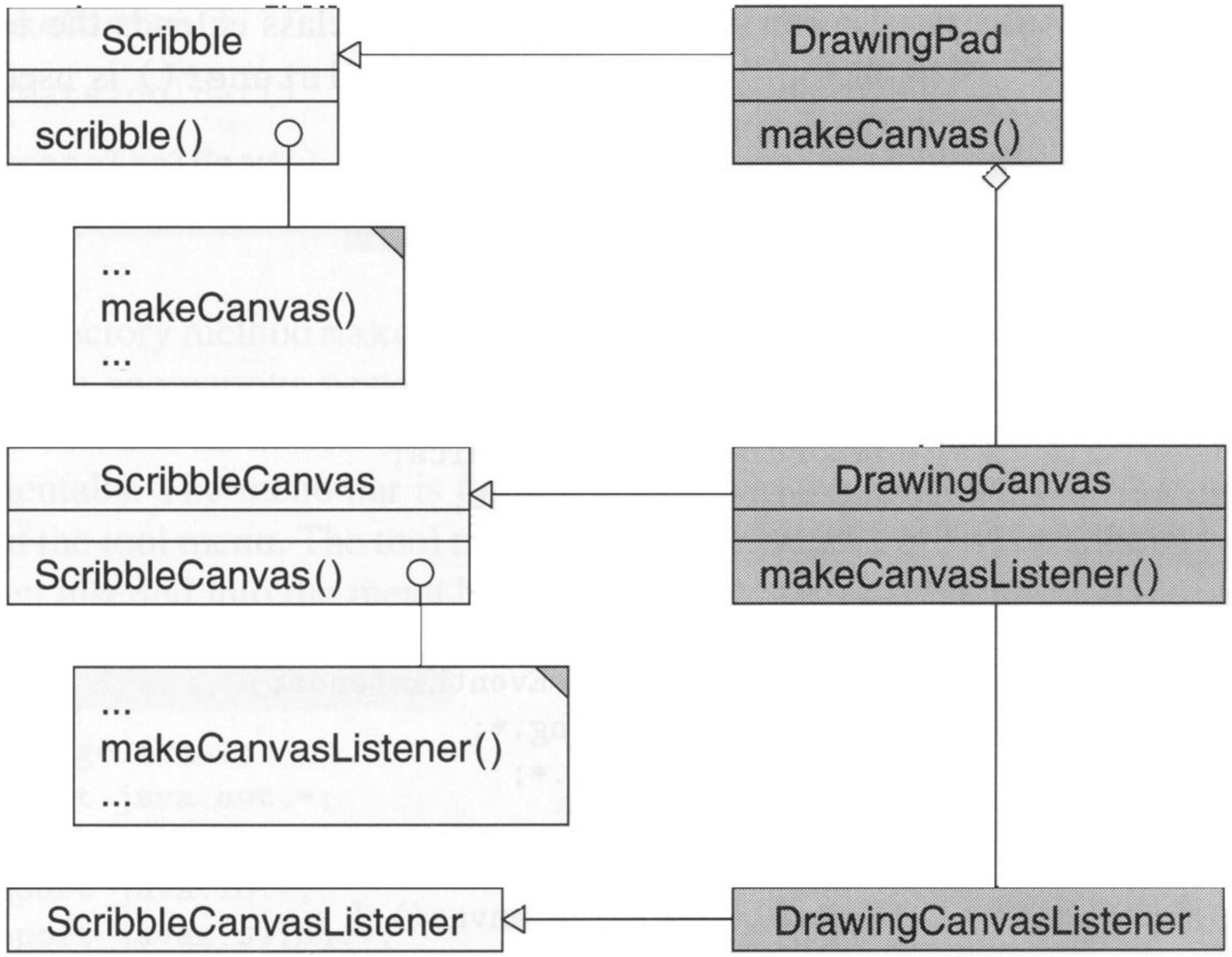
- *Creator* (e.g., `Scribble`), which defines one or more factory methods (e.g. `makeCanvasListener()`) that create abstract products, that is, objects of type *Product*. The *Creator* may provide a default implementation (e.g., the implementation of the `makeCanvasListener()` method in the `Scribble` class) and may call factory methods to create *Product* objects (e.g., invocation of the `makeCanvasListener()` method in the constructor of the `Scribble` class).
- *ConcreteCreator* (e.g., `DrawingPad`), which overrides the factory method to return an instance of a `ConcreteProduct` (e.g., implementation of the `makeCanvasListener()` method in the `DrawingPad` class).

9.5.6 Design Pattern : Factory Method

- The participants of the Factory Method design pattern
  - Product (e.g. EventListener), which defines the interface of the objects the factory method creates
  - ConcreteProduct(e.g. ScribbleCanvasListener and ToolListener), which implements the Product interface (p 427)
  - Creator(e.g. Scribble, p429), which declares the factory method(e.g. makeCanvasListener()) that returns an object of type Product . The creator <u>may provide a default implementation</u>(e.g. the implementation of the makeCanvasListener() method in the Scribble class)
  - ConcreteCreator(e.g., DrawingPad) <u>override</u> the factory method …makeCanvasListener (p.445→ p 446. Line 10, p444)

Figure 9.15

The overall design of the drawing pad—iteration 4.



```
package scribble3;
…………………..
public class ScribbleCanvas extends JPanel {
 public ScribbleCanvas() {
    // calling factory method
    listener = makeCanvasListener();
    addMouseListener((MouseListener) listener);
    addMouseMotionListener((MouseMotionListener) listener);
  }
 // factory method
  protected EventListener makeCanvasListener() {
    return new ScribbleCanvasListener(this);
  }
```

```
package draw1;
…………..

public class DrawingCanvas extends ScribbleCanvas {
 …………..
 // factory method
  protected EventListener makeCanvasListener() {
    return (drawingCanvasListener = new DrawingCanvasListener(this));
  }
  protected DrawingCanvasListener drawingCanvasListener;
}
```

```
 package scribble3;
…………………………..
public class Scribble extends JFrame {
 public Scribble(String title) {
    super(title);
    // calling factory method
    canvas = makeCanvas();
  …………………….
  // factory method
  protected ScribbleCanvas makeCanvas() {
    return new ScribbleCanvas();
  }
 ……….
}
```

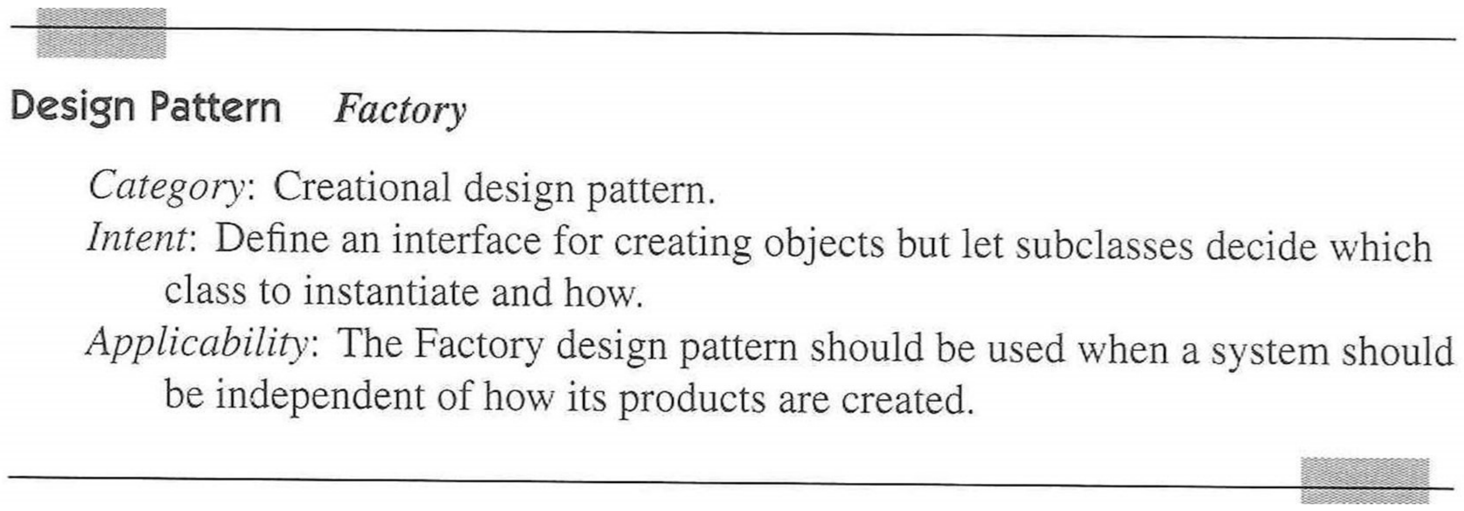
```
 package draw1;
……………………
public class DrawingPad extends Scribble {
………………
 // factory method (Scribble : p. 429,Slide#17)
  protected ScribbleCanvas makeCanvas() {
    return (drawingCanvas = new DrawingCanvas());
  }
 ……………
}
```

9.5.6 Design Pattern : Factory Method

- Factory Method & Factory
  - Factory design pattern (p 296)
    - involves factory class whose sole responsibility is to create objects.
  - Factory Method
    - the creators in the Factory Method pattern are also responsibilities for <u>building a structure</u> using the products created by the factory methods.
    - Factory Method is for a class to defer the creation of certain objects to <u>its subclasses.</u>

(7.4.3 Design Pattern : Factory)

- Design Pattern: Factory

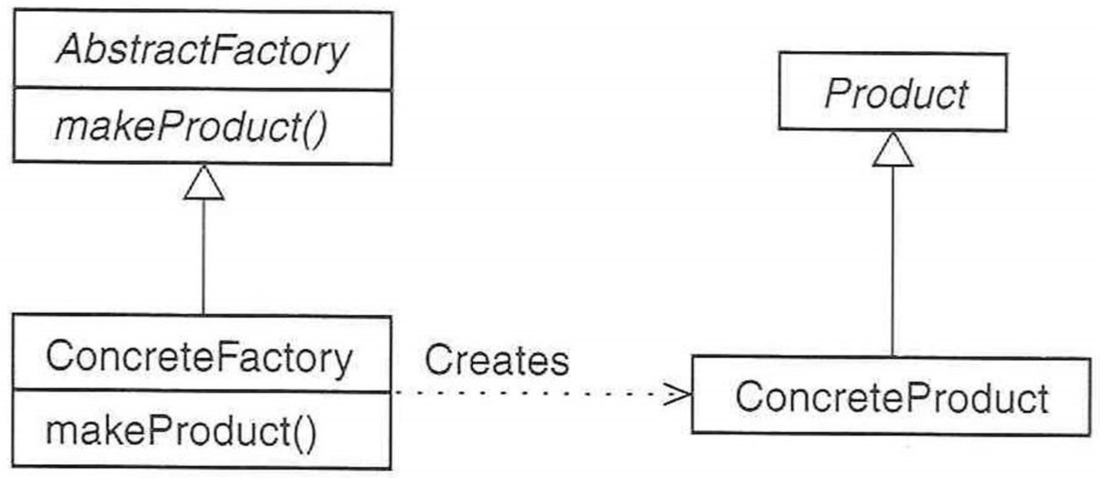


Design Pattern *Factory*

Category: Creational design pattern.

Intent: Define an interface for creating objects but let subclasses decide which class to instantiate and how.

Applicability: The Factory design pattern should be used when a system should be independent of how its products are created.



9.6 Iteration 5 : More Drawing Tools

- Enhance the drawing tool
  - by adding tools for drawing filled ovals and rectangles
  - we <u>refactor</u> the design of the TwoEndsTool(p 440) class to make it more extensible

- 9.6.1 Filled Shapes
  -

```
// Class Draw2.FilledOvalShape

package draw2;

import java.awt.*;
import draw1.*;

public class FilledOvalShape extends OvalShape {

  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.fillOval(x, y, w, h);
  }

}
```

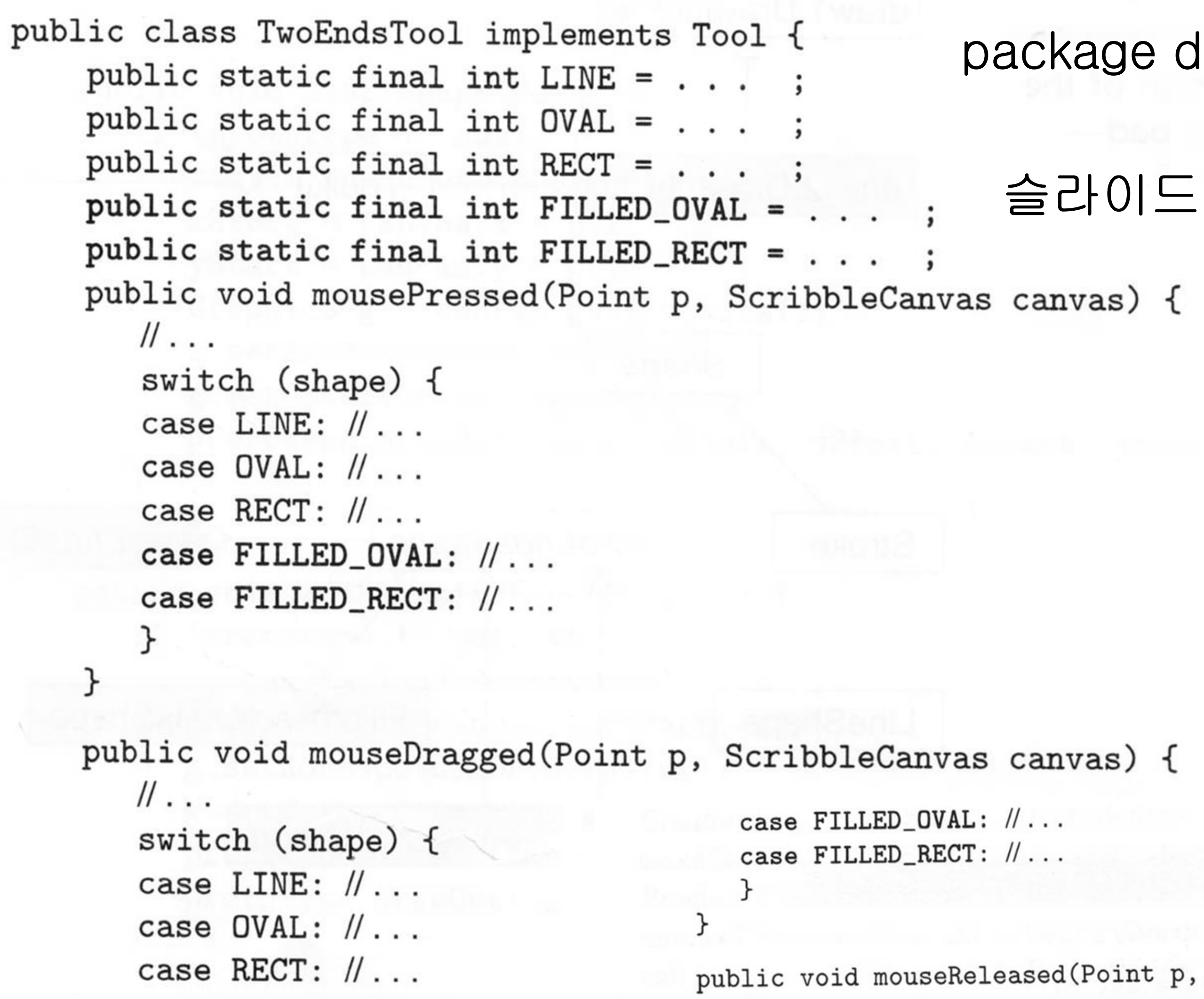
```
// Class draw2.FilledRectangleShape

package draw2;

import java.awt.*;
import draw1.*;

public class FilledRectangleShape extends RectangleShape {

  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.fillRect(x, y, w, h);
  }

}
```

9.6.2 Drawing Filled Shapes

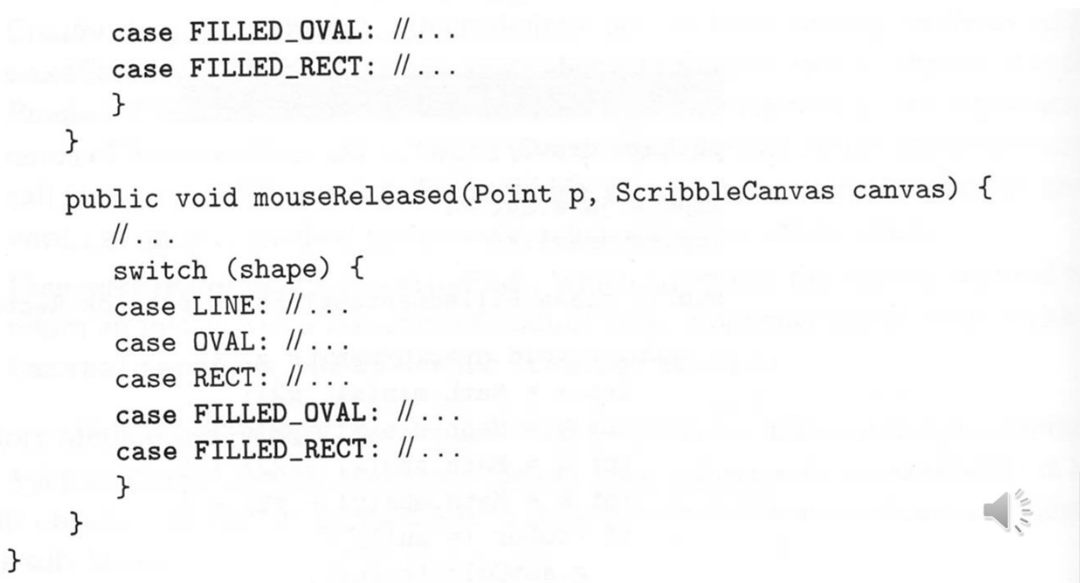
- A Simple ad hoc approach
  - P 449.
  - modification is required for each new shape
  - the relevant code segments for each shape are scattered in three different methods : mousePressed(), mouseDragged() and mouseReleased()

package draw1;

슬라이드 47



```
public class TwoEndsTool implements Tool {
    public static final int LINE = . . .  ;
    public static final int OVAL = . . .  ;
    public static final int RECT = . . .  ;
    public static final int FILLED_OVAL = . . .  ;
    public static final int FILLED_RECT = . . .  ;
    public void mousePressed(Point p, ScribbleCanvas canvas) {
       //...
       switch (shape) {
       case LINE: //...
       case OVAL: //...
       case RECT: //...
       case FILLED_OVAL: //...
       case FILLED_RECT: //...
       }
    }

    public void mouseDragged(Point p, ScribbleCanvas canvas) {
       //...
       switch (shape) {
       case LINE: //...
       case OVAL: //...
       case RECT: //...
```



```
       case FILLED_OVAL: //...
       case FILLED_RECT: //...
       }
    }

    public void mouseReleased(Point p, ScribbleCanvas canvas) {
       //...
       switch (shape) {
       case LINE: //...
       case OVAL: //...
       case RECT: //...
       case FILLED_OVAL: //...
       case FILLED_RECT: //...
       }
    }
}
```

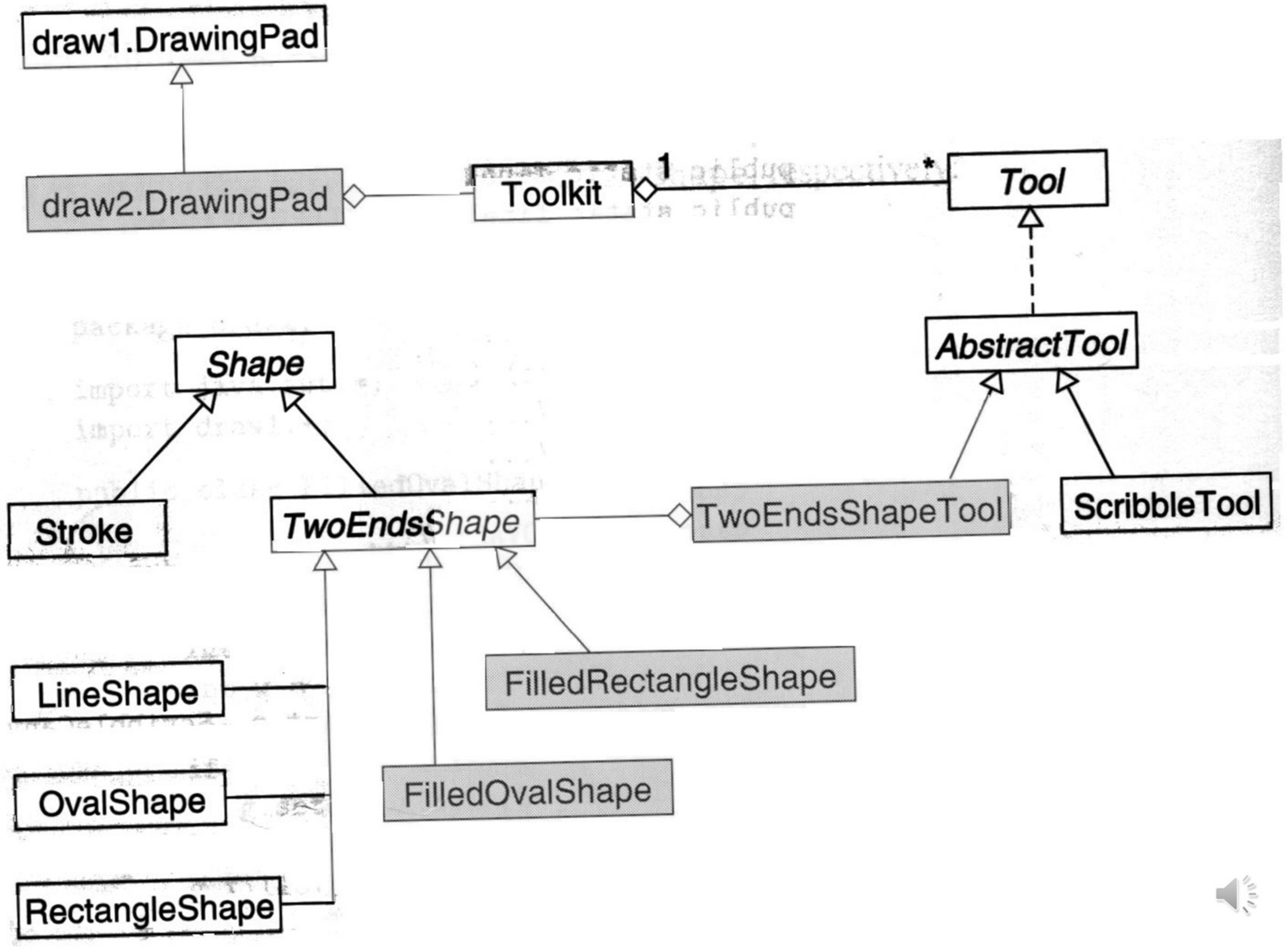
9.6.2 Drawing Filled Shapes

- better design
  - to separate the shape from the tool for drawing that shape using the Strategy design pattern [p. 275]
  - TwoEndsShapeTool
    - instead of using an integer value to indicate the shape to be drawn
    - contains a reference to TwoEndsShape[p.434], which represents abstract strategy.
    - does not contain code that is specific to any particular shape.

Fig 9.16. The design of the drawing pad – iteration 5.

Figure 9.16

The design of the drawing pad—iteration 5.



√

The design of the drawing pad – 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.

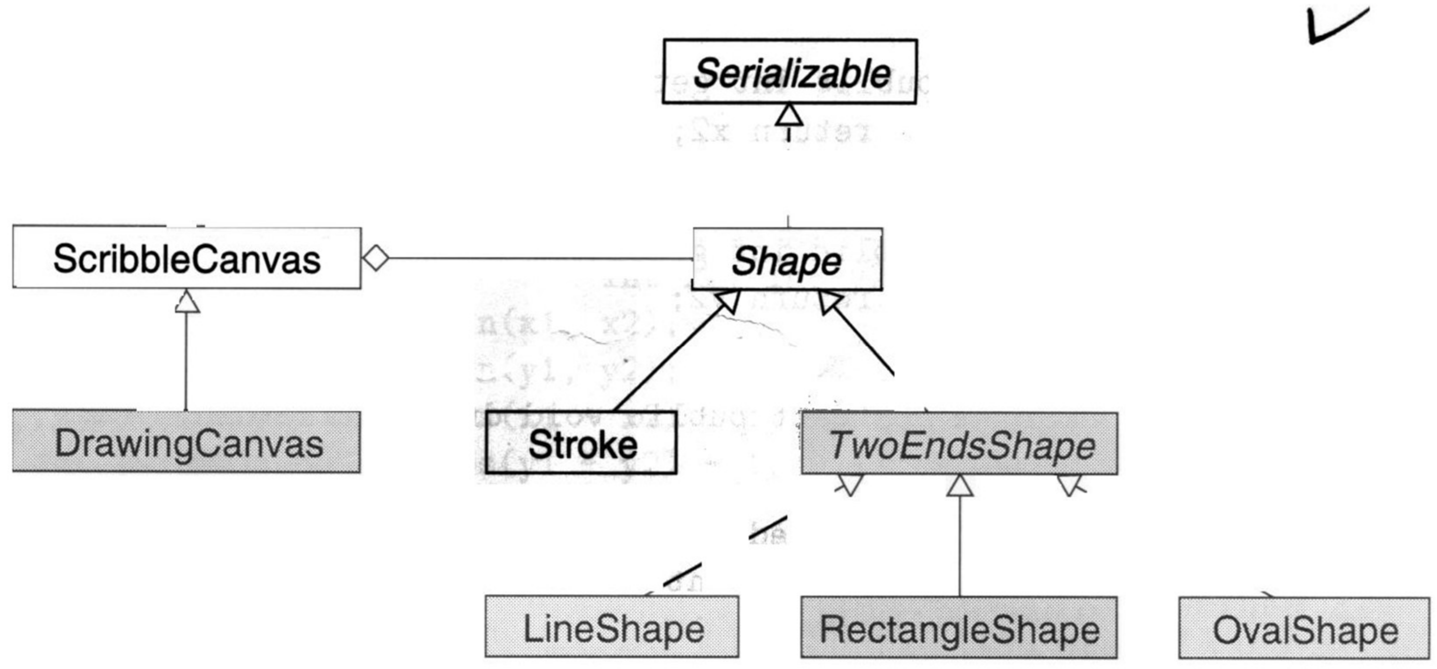
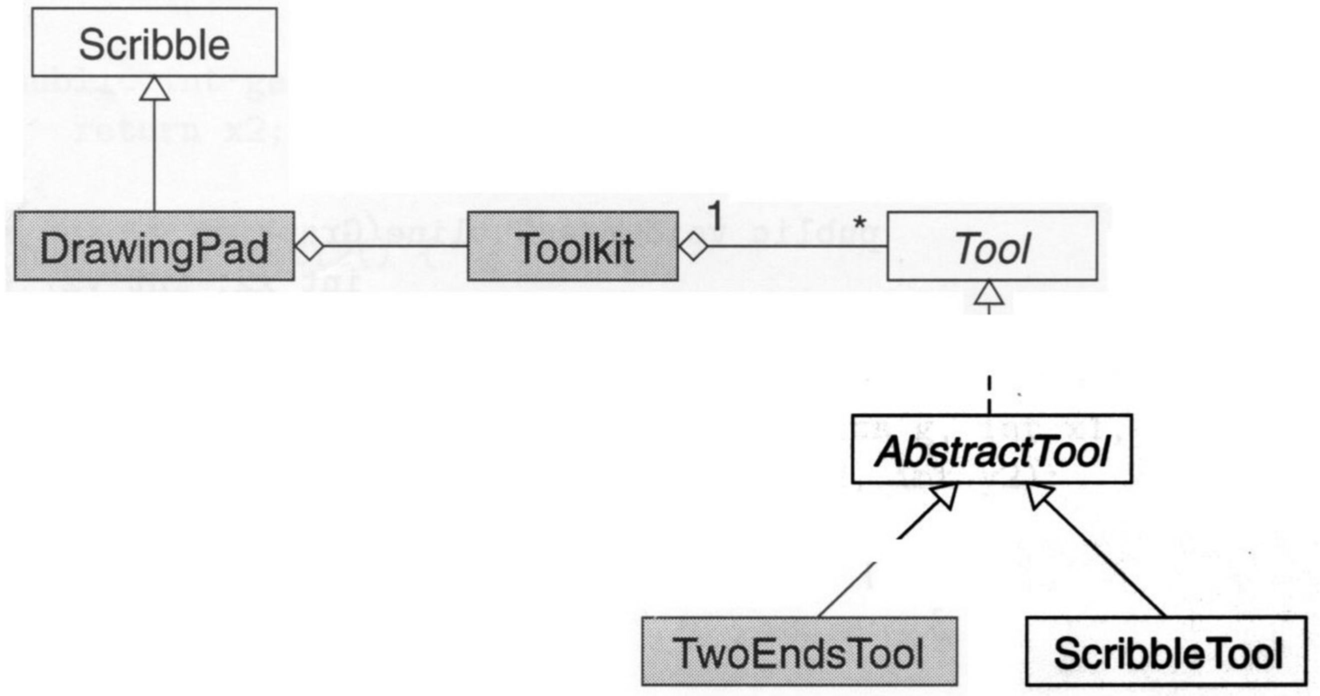


Fig 9.14. The design of the drawing pad – the tools (P. 436)

Figure 9.14

The design of the drawing pad—the tools.



V

```
// Class draw2.TwoEndsShapeTool
// prototype object : serves as a representative of the shape
package draw2;
import java.awt.*;
import scribble3.*;
import draw1.*;

public class TwoEndsShapeTool extends AbstractTool {
  // object prototype : servers as a representative of the shape to be drawn
 //  delegate actions specific to each shape to the prototype object
  public TwoEndsShapeTool(ScribbleCanvas canvas, String name, TwoEndsShape
prototype) {
    super(canvas, name);
    this.prototype = prototype;
  }

  public void startShape(Point p) {
    if (prototype != null) {
      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);
      prototype.drawOutline(g, xStart, yStart, xStart, yStart);
    }
  }
public void addPointToShape(Point p) {
    if (prototype != null &&
            canvas.mouseButtonDown) {
      Graphics g = canvas.getGraphics();
      g.setXORMode(Color.darkGray);
      g.setColor(Color.lightGray);
      prototype.drawOutline(g, xStart, yStart, canvas.x, canvas.y);
      prototype.drawOutline(g, xStart, yStart, p.x, p.y);
    }
  }
  public void endShape(Point p) {
    canvas.mouseButtonDown = false;
    if (prototype != null) {
      try {
            TwoEndsShape newShape = (TwoEndsShape) prototype.clone();
            newShape.setColor(canvas.getCurColor());
            newShape.setEnds(xStart, yStart, p.x, p.y);
            canvas.addShape(newShape);
      } catch (CloneNotSupportedException e) {}
      Graphics g = canvas.getGraphics();
      g.setPaintMode();
      canvas.repaint();
    }
  }
  protected int xStart, yStart;
  protected TwoEndsShape prototype;
}
```

```
// 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;
    }
  }
```

9.6.3 The Application

- The main application class draw2.DrawingPad
  - overrides the initTools() to include two addional drawing tools for filled ovals and filled rectangle.

```java
package draw2;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
import scribble3.*;
import draw1.*;

public class DrawingPad extends draw1.DrawingPad {

  public DrawingPad(String title) {
    super(title);
  }

  protected void initTools() {
    toolkit = new ToolKit();
    toolkit.addTool(new ScribbleTool(canvas, "Scribble"));
    toolkit.addTool(new TwoEndsShapeTool(canvas, "Line", new LineShape()));
    toolkit.addTool(new TwoEndsShapeTool(canvas, "Oval", new OvalShape()));
    toolkit.addTool(new TwoEndsShapeTool(canvas, "Rect", new
RectangleShape()));
    toolkit.addTool(new TwoEndsShapeTool(canvas, "Filled Oval", new
FilledOvalShape()));
    toolkit.addTool(new TwoEndsShapeTool(canvas, "Filled Rect", new
FilledRectangleShape()));
    drawingCanvas.setTool(toolkit.getTool(0));
  }
  public static void main(String[] args) {
    JFrame frame = new draw2.DrawingPad("Drawing Pad");
    frame.setSize(width, height);
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    frame.setLocation(screenSize.width/2 - width/2,
                              screenSize.height/2 - height/2);
    frame.show();
  }

}
```

9.7 Iteration 6 :The Text Tool

- Enhance the drawing pad by adding a tool for typing text using the keyboard.
  - Fig 9.17. The Drawing Pad – Iteration 6.
  - Fig 9.18. The design of the drawing pad – iteration 6.
  - Key issues
    - handling keyboard input
    - keyboard focus

Fig 9.17. The Drawing Pad – Iteration 6.

Figure 9.17

The drawing pad—iteration 6.

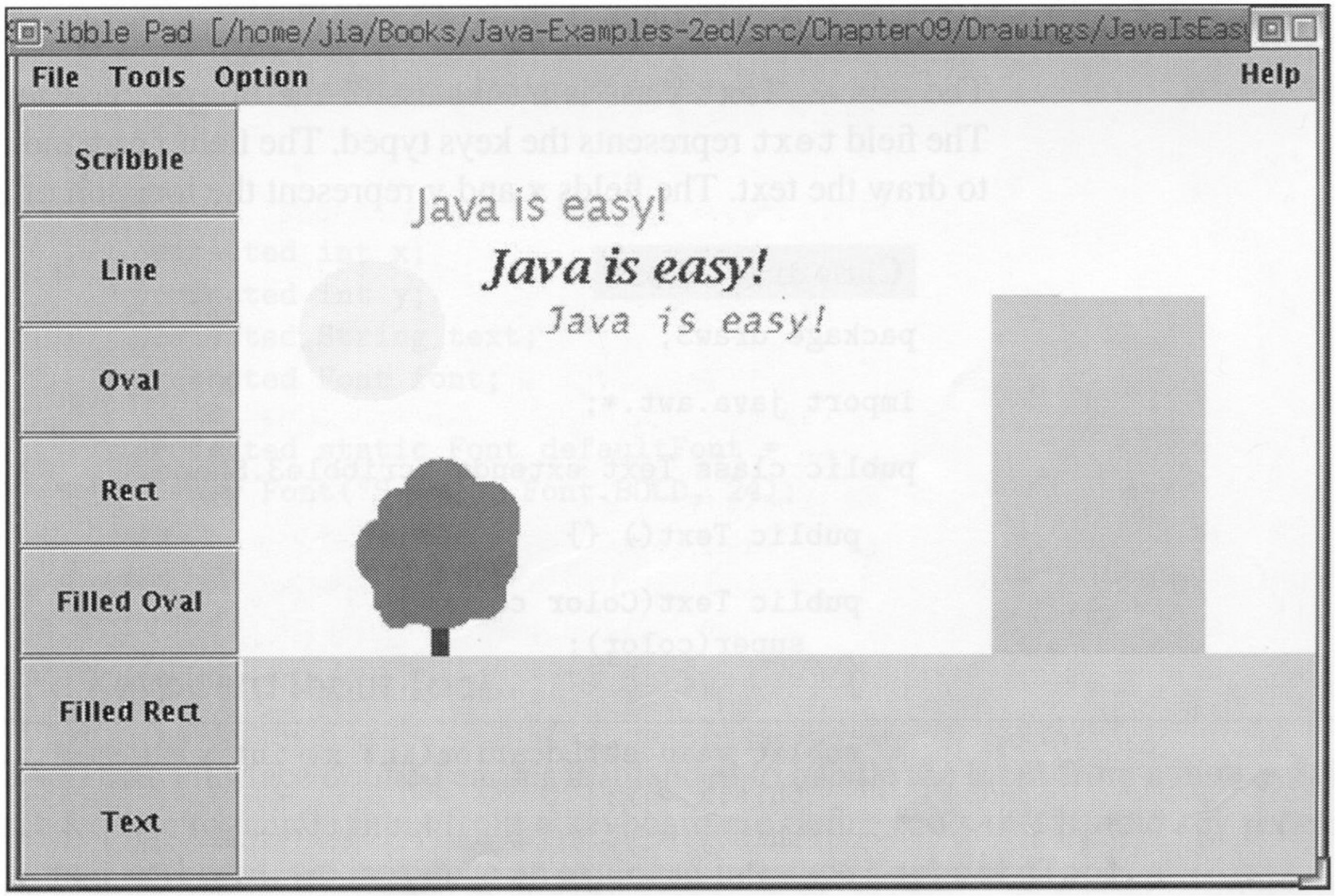
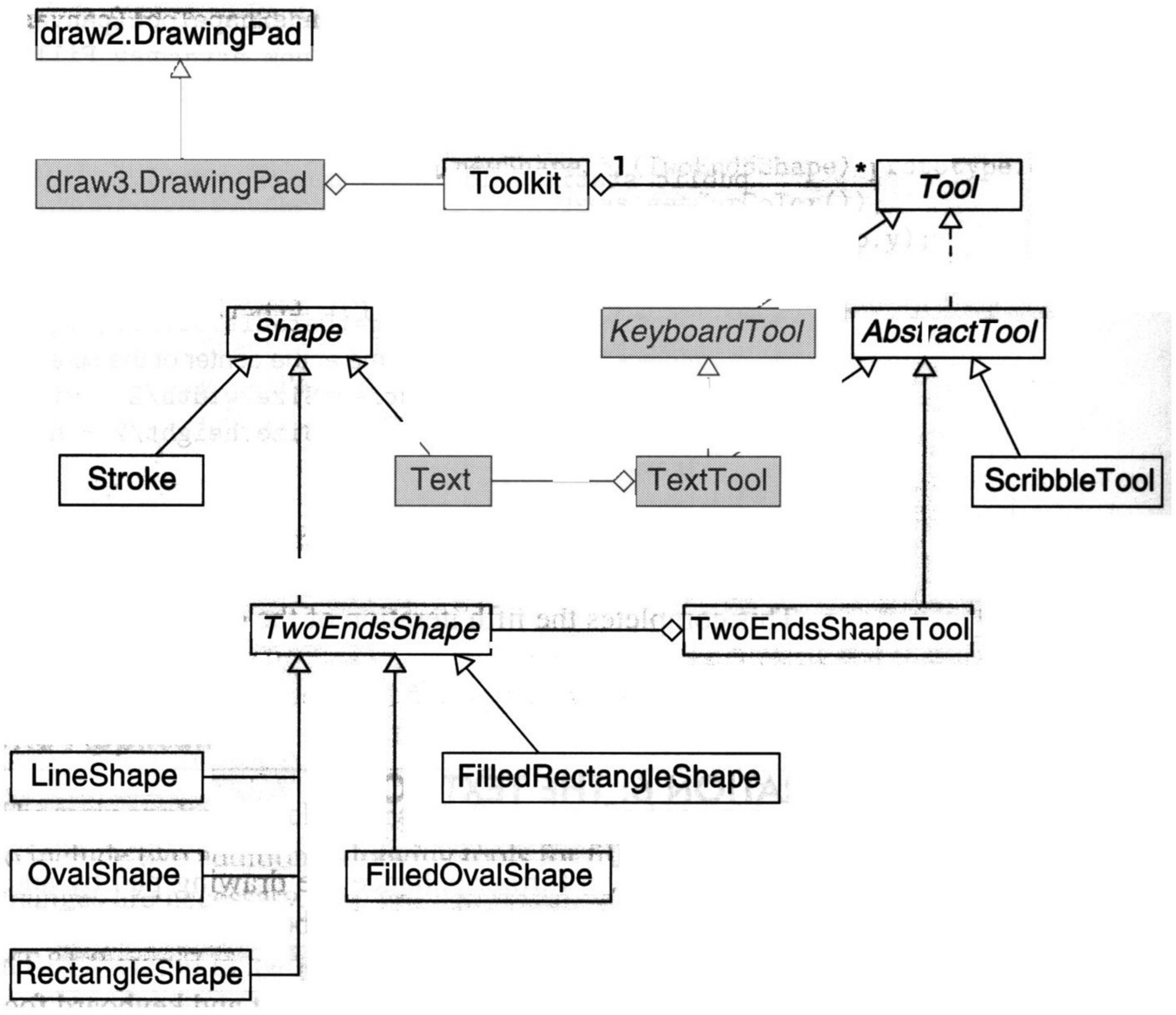


Fig 9.18. The design of the drawing pad – iteration 6.

Figure 9.18

The design of the drawing pad— iteration 6.



9.7.1. The Text Shape

- 9.7.1. The Text Shape
  - the field text : the keys typed.
  - font
  - x, y

```
// Class draw3.Text : a subclass of the Shape
// text field : the keys typed

package draw3;
import java.awt.*;

public class Text extends scribble3.Shape {

  public Text() {}

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

  public void setLocation(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public int getX() {
    return x;
  }

  public int getY() {
    return y;
  }

  public void setText(String text) {
    this.text = text;
  }
 public String getText() {
    return text;
  }

  public Font getFont() {
    return font;
  }

  public void setFont(Font font) {
    this.font = font;
  }

  public void draw(Graphics g) {
    if (text != null) {
      if (color != null) {
                g.setColor(color);
      }
      if (font != null) {
                g.setFont(font);
      } else {
                g.setFont(defaultFont);
      }
      g.drawString(text, x, y);
    }
  }

  protected int x;
  protected int y;
  protected String text;
  protected Font font;

  protected static Font defaultFont = new
Font("Serif", Font.BOLD, 24);

}
```

9.7.2 The Keyboard Input Tool

- Extended interface KeyboardTool

```
package draw3;

import scribble3.Tool;

public interface KeyboardTool extends Tool {

  public void addCharToShape(char c);

}
```

9.7.2 The Keyboard Input Tool

- The TextTool class
  - represents <u>a concrete tool</u> that handles input from the keyboard
  - text field : use StringBuffer instead of String
  - behavior
    - a mouse press indicates the position in the canvas where the text will be displayed.
    - whenever a key is pressed, the corresponding character is appended to text, and it is drawn in the canvas.

```
// Class draw3.TextTool
package draw3;

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

public class TextTool extends AbstractTool implements KeyboardTool {

  public TextTool(ScribbleCanvas canvas, String name) {
    super(canvas, name);
    text = new StringBuffer();
  }

  public void startShape(Point p) {
    text.delete(0, text.length());
    curShape = new Text();
    curShape.setColor(canvas.getCurColor());
    curShape.setLocation(p.x, p.y);
    if (canvas instanceof KeyboardDrawingCanvas) {
      curShape.setFont(((KeyboardDrawingCanvas) canvas).getFont());
    }
    canvas.addShape(curShape);
  }
public void addCharToShape(char c) {
    text.append(c);
    curShape.setText(text.toString());
    canvas.repaint();
  }

  public void addPointToShape(Point p) {}
  public void endShape(Point p) {}

  protected StringBuffer text;
  protected Text curShape;

}
```

9.7.2 The Keyboard Input Tool

- The EventListener of the canvas must be extended.
  - KeyListener : the listener interface

```
// Class draw3.KeyDrawingCanvasListener
package draw3;

import java.awt.*;
import java.awt.event.*;
import draw1.*;

public class KeyboardDrawingCanvasListener extends DrawingCanvasListener
implements KeyListener {
  public KeyboardDrawingCanvasListener(DrawingCanvas canvas) {
    super(canvas);
  }

  public void keyPressed(KeyEvent e) {
    if (tool instanceof KeyboardTool) {
      KeyboardTool keyboardTool = (KeyboardTool) tool;
      keyboardTool.addCharToShape((char) e.getKeyChar());
    }
  }

  public void keyReleased(KeyEvent e) {}
  public void keyTyped(KeyEvent e) {}
  public void mouseClicked(MouseEvent e) {
    canvas.requestFocus();
  }
}
```

9.7.2 The Keyboard Input Tool

- Keyboard focus
  - only one component will receive keyboard input at any moment
  - the component having the keyboard focus is determined by a focus manager.
  - isFocusable()

```
 // Class draw3.KeyDrawingCanvas
package draw3;

import java.awt.*;
import java.awt.event.*;
import java.util.EventListener;
import draw1.*;

public class KeyboardDrawingCanvas extends DrawingCanvas {

  public KeyboardDrawingCanvas() {
    addKeyListener((KeyListener) listener);
    font = new Font(fontFamily, fontStyle, fontSize);
  }

  public Font getFont() {
    return font;
  }

  public String getFontFamily() {
    return fontFamily;
  }
```

```
public void setFontFamily(String fontFamily) {
  if (fontFamily != null &&
          !fontFamily.equals(this.fontFamily)) {
    this.fontFamily =fontFamily;
    font = new Font(fontFamily, fontStyle, fontSize);
  }
}

public int getFontSize() {
  return fontSize;
}

public void setFontSize(int fontSize) {
  if (fontSize > 0 &&
          fontSize != this.fontSize) {
    this.fontSize = fontSize;
    font = new Font(fontFamily, fontStyle, fontSize);
  }
}

public int getFontStyle() {
  return fontStyle;
}
```

```
 public void setFontStyle(int fontStyle) {
    if (fontStyle != this.fontStyle) {
      this.fontStyle = fontStyle;
      font = new Font(fontFamily, fontStyle, fontSize);
    }
  }

  // necessary for keyboard input
  // public boolean isFocusTraversable() { // pre 1.4
  public boolean isFocusable() { // 1.4
    return true;
  }

  // factory method
  protected EventListener makeCanvasListener() {
    return (drawingCanvasListener = new
KeyboardDrawingCanvasListener(this));
  }
  protected String fontFamily = "Serif";
  protected int fontSize = 24;
  protected int fontStyle = Font.PLAIN;
  protected Font font;

}
```

9.7.3 The Font Option Menu

- Draw3.DrawingPad
  - new text tool to toolkit
  - add a cascading menu to the menu bar for selecting the font family, font size, and font style.

```
package draw3;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
import draw2.*;
import scribble3.*;

public class DrawingPad extends draw2.DrawingPad {

  public DrawingPad(String title) {
    super(title);
    JMenu optionMenu = menuBar.getMenu(2);
    addFontOptions(optionMenu);
  }
  // factory method
  protected ScribbleCanvas makeCanvas() {
    return (drawingCanvas = keyboardDrawingCanvas = new
KeyboardDrawingCanvas());
  }

  protected void initTools() {
    super.initTools();
    toolkit.addTool(new TextTool(canvas, "Text"));
  }
```

```
protected void addFontOptions(JMenu optionMenu) {
  String[] fontFamilyNames = {
    "Serif",
    "Sans-serif",
    "Monospaced",
    "Dialog",
    "DialogInput"
  };

  int[] fontSizes = {
    8, 10, 12, 16, 20, 24, 28, 32, 40, 48, 64
  };

  String[] fontStyleNames = {
    "plain",
    "bold",
    "italic",
    "bold italic"
  };
```

```
int i;
   ActionListener fontFamilyAction = new ActionListener() {
           public void actionPerformed(ActionEvent event) {
             Object source = event.getSource();
             if (source instanceof JCheckBoxMenuItem) {
               JCheckBoxMenuItem mi = (JCheckBoxMenuItem) source;
               String name = mi.getText();
               keyboardDrawingCanvas.setFontFamily(name);
             }
           }
     };

 JMenu fontFamilyMenu = new JMenu("Font family");
   ButtonGroup group = new ButtonGroup();
   for (i = 0; i < fontFamilyNames.length; i++) {
     JCheckBoxMenuItem mi = new JCheckBoxMenuItem(fontFamilyNames[i]);
     fontFamilyMenu.add(mi);
     mi.addActionListener(fontFamilyAction);
     group.add(mi);
   }
   optionMenu.add(fontFamilyMenu);
```

```
ActionListener fontSizeAction = new ActionListener() {
      public void actionPerformed(ActionEvent event) {
        Object source = event.getSource();
        if (source instanceof JCheckBoxMenuItem) {
          JCheckBoxMenuItem mi = (JCheckBoxMenuItem) source;
          String size = mi.getText();
          try {
            keyboardDrawingCanvas.setFontSize(Integer.parseInt(size));

          } catch (NumberFormatException e) {}
        }
      }
   };
  JMenu fontSizeMenu = new JMenu("Font size");
  group = new ButtonGroup();
  for (i = 0; i < fontSizes.length; i++) {
    JCheckBoxMenuItem mi = new
JCheckBoxMenuItem(Integer.toString(fontSizes[i]));
    fontSizeMenu.add(mi);
    mi.addActionListener(fontSizeAction);
    group.add(mi);
  }
  optionMenu.add(fontSizeMenu);
```

```
ActionListener fontStyleAction = new ActionListener() {
          public void actionPerformed(ActionEvent event) {
            Object source = event.getSource();
            if (source instanceof JCheckBoxMenuItem) {
              JCheckBoxMenuItem mi = (JCheckBoxMenuItem) source;
              String styleName = mi.getText();
              int style = Font.PLAIN;
              if (styleName.equals("bold")) {
                style = Font.BOLD;
              } else if (styleName.equals("italic")) {
                style = Font.ITALIC;
              } else if (styleName.equals("bold italic")) {
                style = Font.BOLD | Font.ITALIC;
              }
              keyboardDrawingCanvas.setFontStyle(style);
            }
          }
    };
```

```
JMenu fontStyleMenu = new JMenu("Font style");
    group = new ButtonGroup();
    for (i = 0; i < fontStyleNames.length; i++) {
      JCheckBoxMenuItem mi = new
JCheckBoxMenuItem(fontStyleNames[i]);
      fontStyleMenu.add(mi);
      mi.addActionListener(fontStyleAction);
      group.add(mi);
    }
    optionMenu.add(fontStyleMenu);
  }
protected KeyboardDrawingCanvas keyboardDrawingCanvas;
```

```
public static void main(String[] args) {
   JFrame frame = new draw3.DrawingPad("Drawing Pad");
   frame.setSize(width, height);
   Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
   frame.setLocation(screenSize.width/2 - width/2,
                     screenSize.height/2 - height/2);
   frame.show();
 }

}
```

끝