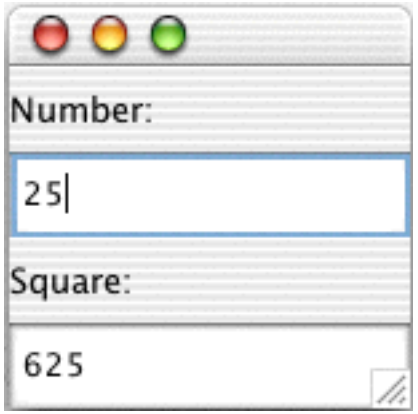
Grafiske brugergrænseflader II

Plan

- MVC (Model View Controller)
  - designmønsteret Observer

- Iterativ udvikling af et tegneprogram
  - muselyttere
  - applet/applikation-idiomet
  - designmønsteret State
  - designmønsteret Factory Method

En grafisk brugergrænseflade til beregning af kvadratet på et tal



```
public class SquareFrame extends JFrame implements ActionListener {
    private Label numberLabel, squareLabel;
    private TextField numberField, squareField;

    public SquareFrame() {
        numberLabel = new Label("Number:");
        numberField = new TextField();
        squareLabel = new Label("Square:");
        squareField = new TextField();
        squareField.setEditable(false);
        Container pane = getContentPane();
        pane.setLayout(new GridLayout(4, 1));
        pane.add(numberLabel);
        pane.add(numberField);
        pane.add(squareLabel);
        pane.add(squareField);
        numberField.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        int number = Integer.parseInt(numberField.getText());
        squareField.setText(Integer.toString(number * number));
    }
}
```

Model-View-Controller (MVC)

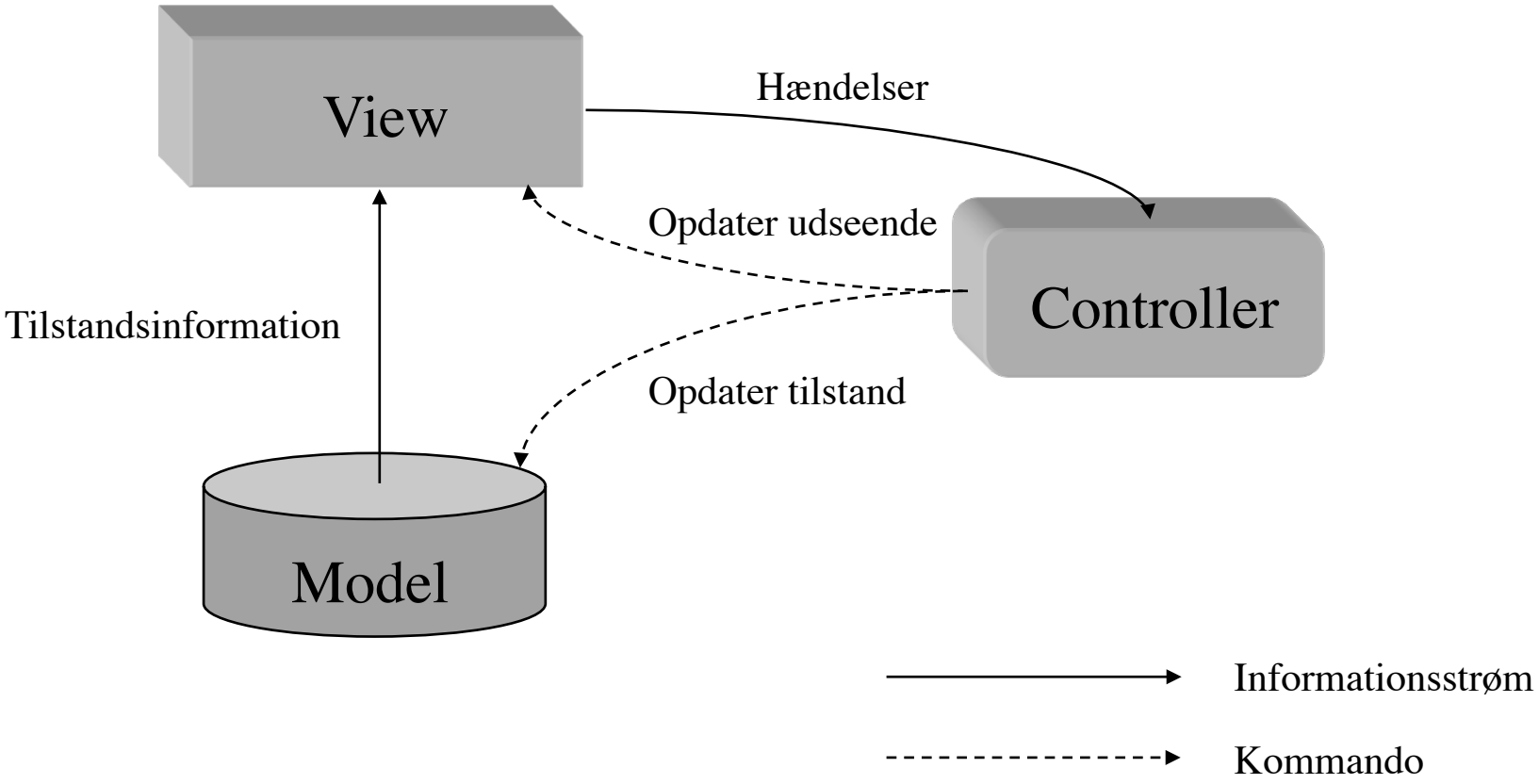
- et arkitektonisk mønster

For at gøre udskiftning af en brugergrænseflade simpel, bør brugergrænsefladen have en løs kobling med de øvrige dele af programmet.

Den fundamentale ide i MVC er at adskille data (*model*) fra deres grafiske præsentation (*view*).

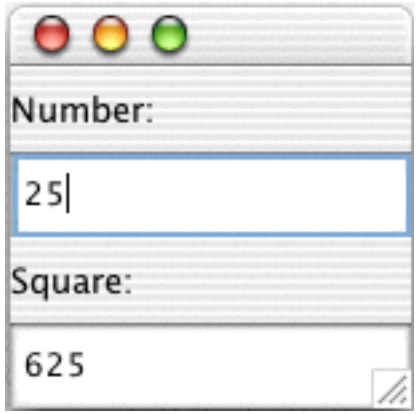
En brugergrænseflades *controller* tager sig af brugerens input (hændelser). Typisk omfatter dette ændringer i modellen.

MVC-arkitekturen



Eksempel på anvendelse af MVC

(et GUI-program, der beregner kvadratet på et tal)



	Metoder:
Model:	`getNumber(), getSquare(), setNumber()`
View:	`getNumber(), update()`
Controller:	`actionPerformed()`

Model

```
public class Model {
    int getNumber() {
        return number;
    }

    int getSquare() {
        return number * number;
    }

    void setNumber(int number) {
        this.number = number;
    }

    private int number;
}
```

Lav model først. Kan testes uafhængigt af brugergrænsefladen.

View

```
public class View extends JFrame {
    public View(Model model) {
        this.model = model;
        numberLabel = new Label("Number:");
        numberField = new JTextField();
        ...
        numberField.addActionListener(new Controller(model, this));
    }

    public int getNumber() {
        return Integer.parseInt(numberField.getText());
    }

    public void update() {
        squareField.setText(Integer.toString(model.getSquare()));
    }

    private Model model;
    private JTextField numberField, squareField;
    private JLabel numberLabel, squareLabel;
}
```

Controller

```
public class Controller implements ActionListener {
    public Controller(Model model, View view) {
        this.model = model;
        this.view = view;
    }

    public void actionPerformed(ActionEvent e) {
        model.setNumber(view.getNumber());
        view.update();
    }

    private Model model;
    private View view;
}
```

Et hovedprogram

```
public class MVC {
    public static void main(String args[]) {
        Model model = new Model();
        View view = new View(model);
        view.setSize(150, 150);
        view.setVisible(true);
        view.setDefaultCloseOperation(view.EXIT_ON_CLOSE);
    }
}
```

`Observer` og `Observable`

Java understøtter brugen af MVC-konceptet igennem grænsefladen `Observer` og klassen `Observable`.

Et `Observable`-objekt har en metode, `addObserver`, som benyttes til at registrere dets `Observer`-objekter.

`Observer`-objekterne kan underrettes om ændringer ved kald af metoden `notifyObservers`. Disse `Observer`-objekter vil da få kaldt deres `update`-metode.

Observer og Observable

java.util.*

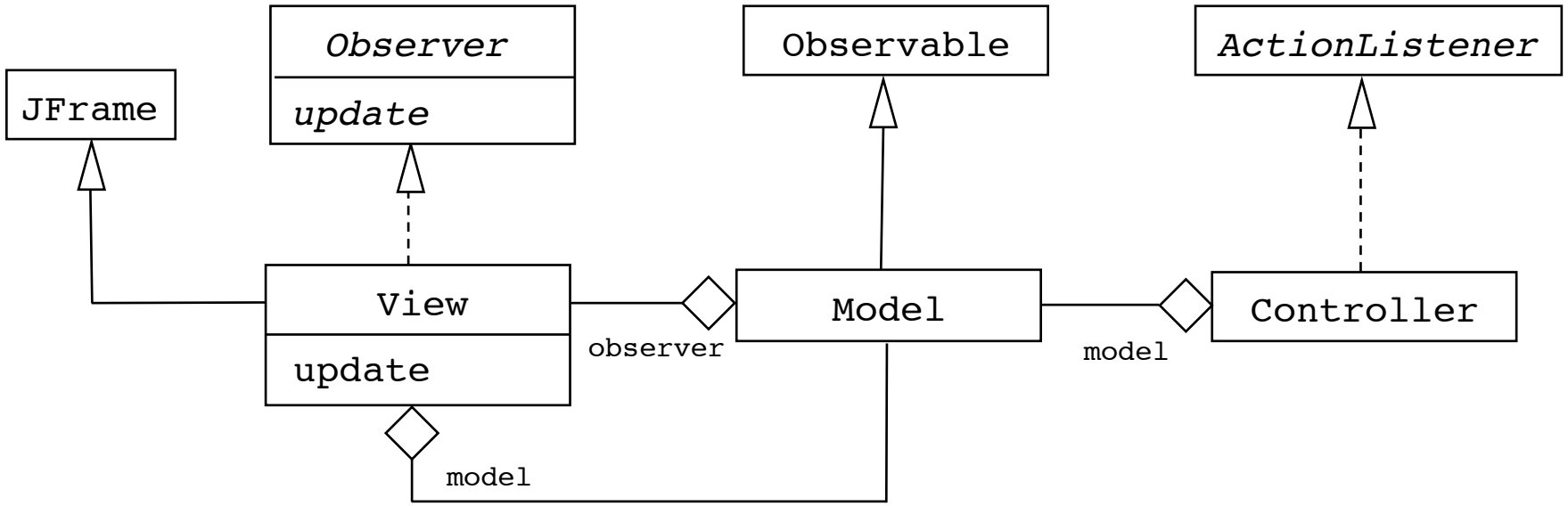
```
public interface Observer {
    void update(Observable obj, Object arg);
}
```

```
public class Observable {
    public void addObserver(Observer obj);
    public void setChanged();
    public void notifyObservers();
    public void notifyObservers(Object arg);
}
```

Anvendelse af `Observer/Observable`

Model:	`extends` **`Observable`**
View:	`implements` **`Observer`**
Controller:	`implements` **`ActionListener`**

Programstruktur



Model

```
public class Model extends Observable {
    public int getNumber() {
        return number;
    }

    public int getSquare() {
        return number * number;
    }

    public int setNumber(int number) {
        this.number = number;
        setChanged();
        notifyObservers();
    }

    private int number;
}
```

View

```
public class View extends JFrame implements Observer {
    public View(Model model) {
        this.model = model;
        model.addObserver(this);
        numberField = new JTextField();

        ...

        numberField.addActionListener(new Controller(model));
    }

    public void update(Observable obj, Object arg) {
        squareField.setText(Integer.toString(model.getSquare()));
    }

    private Model model;
    private TextField numberField, squareField;
    private Label numberLabel, squareLabel;
}
```

Controller

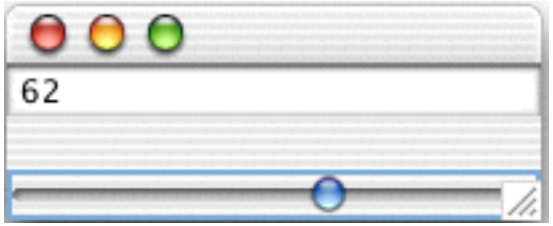
```
public class Controller implements ActionListener {
    private Model model;

    public Controller(Model model) {
        this.model = model;
    }

    public void actionPerformed(ActionEvent e) {
        JTextField numberField = (JTextField) e.getSource();
        model.setNumber(Integer.parseInt(numberField.getText()));
    }
}
```

View og Controller i en og samme klasse
(model-delegate arkitektur)

```
public class View extends JFrame implements Observer, ActionListener {
    public View(Model model) {
        ...
        numberField.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        model.setNumber(Integer.parseInt(numberField.getText()));
    }

    public void update(Observable obj, Object arg) {
        squareField.setText(Integer.toString(model.getSquare()));
    }

    private Model model;
    private TextField numberField, squareField;
    private Label numberLabel, squareLabel;
}
```

Eksempel på anvendelse af model-delegate

(visning af et heltal med to views)




```
ValueModel      extends Observable
TextDelegate    implements Observer, ActionListener
SliderDelegate  implements Observer, ChangeListener
```

ValueModel

```
class ValueModel extends Observable {
    private int value;

    public void setValue(int value) {
        this.value = value;
        setChanged();
        notifyObservers();
    }

    public int getValue() {
        return value;
    }
}
```

TextDelegate

```
class TextDelegate extends JTextField
                   implements Observer, ActionListener {
    private ValueModel model;

    public TextDelegate(ValueModel model) {
        this.model = model;
        model.addObserver(this);
        addActionListener(this);
    }

    public void actionPerformed(ActionEvent evt) {
        model.setValue(Integer.parseInt(getText()));
    }

    public void update(Observable obj, Object arg) {
        if (obj == model)
            setText(Integer.toString(model.getValue()));
    }
}
```

SliderDelegate

```
class SliderDelegate extends JSlider
                     implements Observer, ChangeListener {
    private ValueModel model;

    public SliderDelegate(ValueModel model) {
        super(0, 100, model.getValue());
        this.model = model;
        model.addObserver(this);
        addChangeListener(this);
    }

    public void stateChanged(ChangeEvent evt) {
        model.setValue(getValue());
    }

    public void update(Observable obj, Object arg) {
         if (obj == model)
             setValue(model.getValue());
    }
}
```

MDFrame

```
public class MDFrame extends JFrame {
    public MDFrame() {
        ValueModel model = new ValueModel();
        Container pane = getContentPane();
        pane.add(new TextDelegate(model),   BorderLayout.NORTH);
        pane.add(new SliderDelegate(model), BorderLayout.SOUTH);
        setSize(200, 80);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
        new MDFrame();
    }
}
```

Designmønsteret Observer

Kategori:

Adfærdsmæssigt designmønster

Hensigt:

At definere en en-til-mange-relation imellem objekter, således at når et objekt ændrer tilstand, vil alle dets afhængige objekter blive notificeret og opdateret automatisk

Anvendelse:

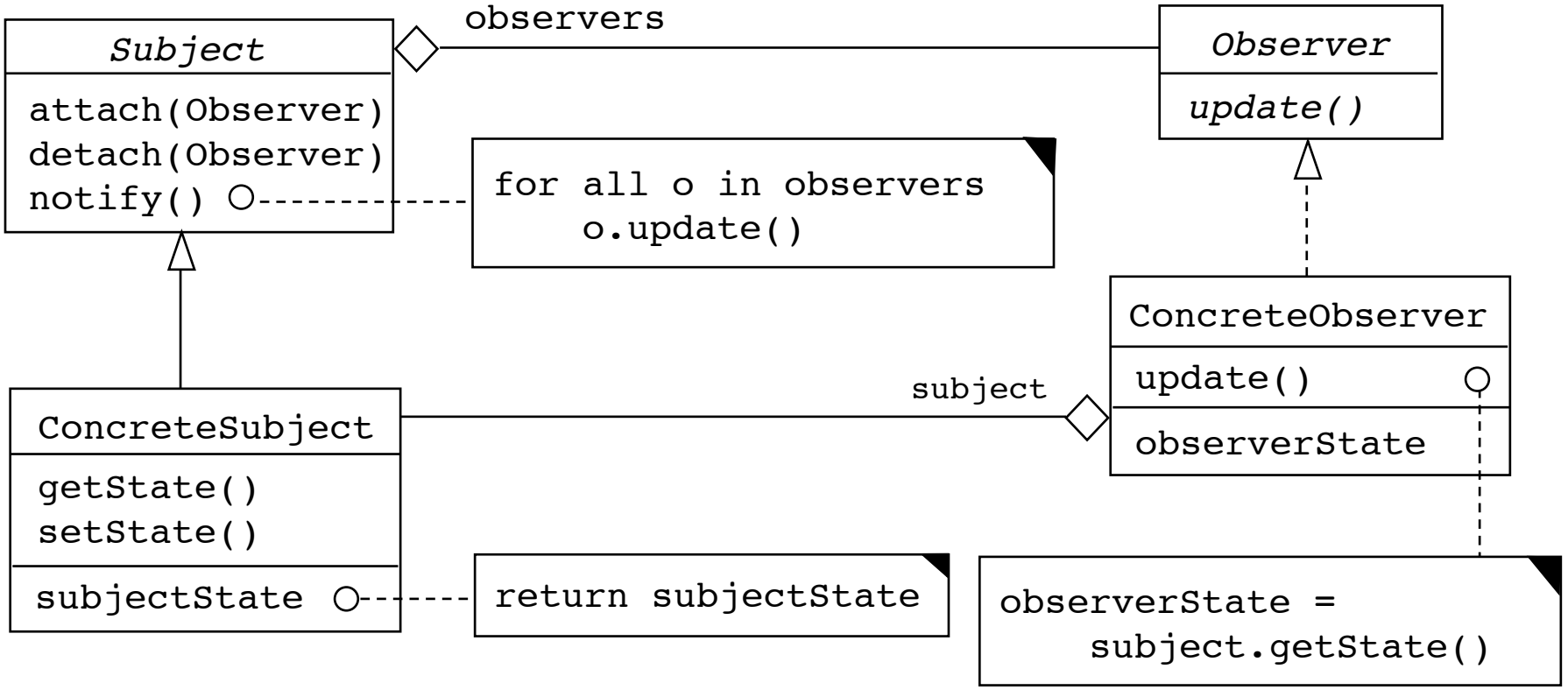
- Når en abstraktion har to sider, hvoraf den ene er afhængig af den anden
- Når en ændring i et objekt kræver ændringer i andre objekter, og deres antal er ukendt
- Når et objekt skal kunne notificere andre objekter uden at kende dem

Andre navne:

Publish-Subscribe

Designmønsteret Observer
(fortsat)

Struktur:



Designmønsteret Observer
(fortsat)

Deltagere:

Subject, der kender sine *Observer*-objekter og definerer en grænseflade for tilknytning og løsrivelse af *Observer*-objekter

Observer, der definerer en grænseflade for objekter, der skal underrettes om ændringer i et *Subject*

ConcreteSubject, der indeholder den tilstand, der har interesse for *ConcreteObserver*-objekter, og som sender besked til sine *Observer*-objekter, når dets tilstand ændres

ConcreteObserver, der har en reference til et *ConcreteSubject*-objekt og implementerer en *update*-metode

Lyttere og kilder

Swings hændelsesmodel benytter Observer-designmønsteret med typificerede notifikations-objekter (`Event`-objekter).

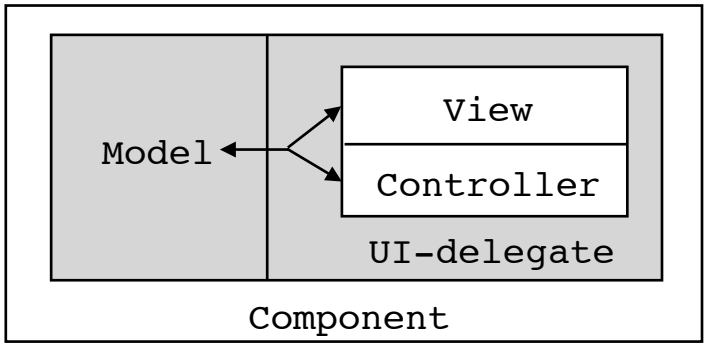
Lytterne er “observers”.

Kilderne er “observable”.

Brug af MVC

- Giver stor fleksibilitet, men på bekostning af større kodemængde
- Ofte slås controller og view sammen (model-delegate arkitektur)

En Swing-komponent:



Iterativ udvikling

- Udviklingsprocessen foregår i iterationer.
- Hver iteration er baseret på resultatet af den forrige iteration.
- Hver iteration resulterer i et fuldt funktionelt mellemprodukt.

Iterativ udvikling af et tegneprogram

1. Et simpelt program til skitsetegning i et givet tegneområde.

2. Muliggør intern og ekstern lagring af tegninger
   samt valg af penfarve.

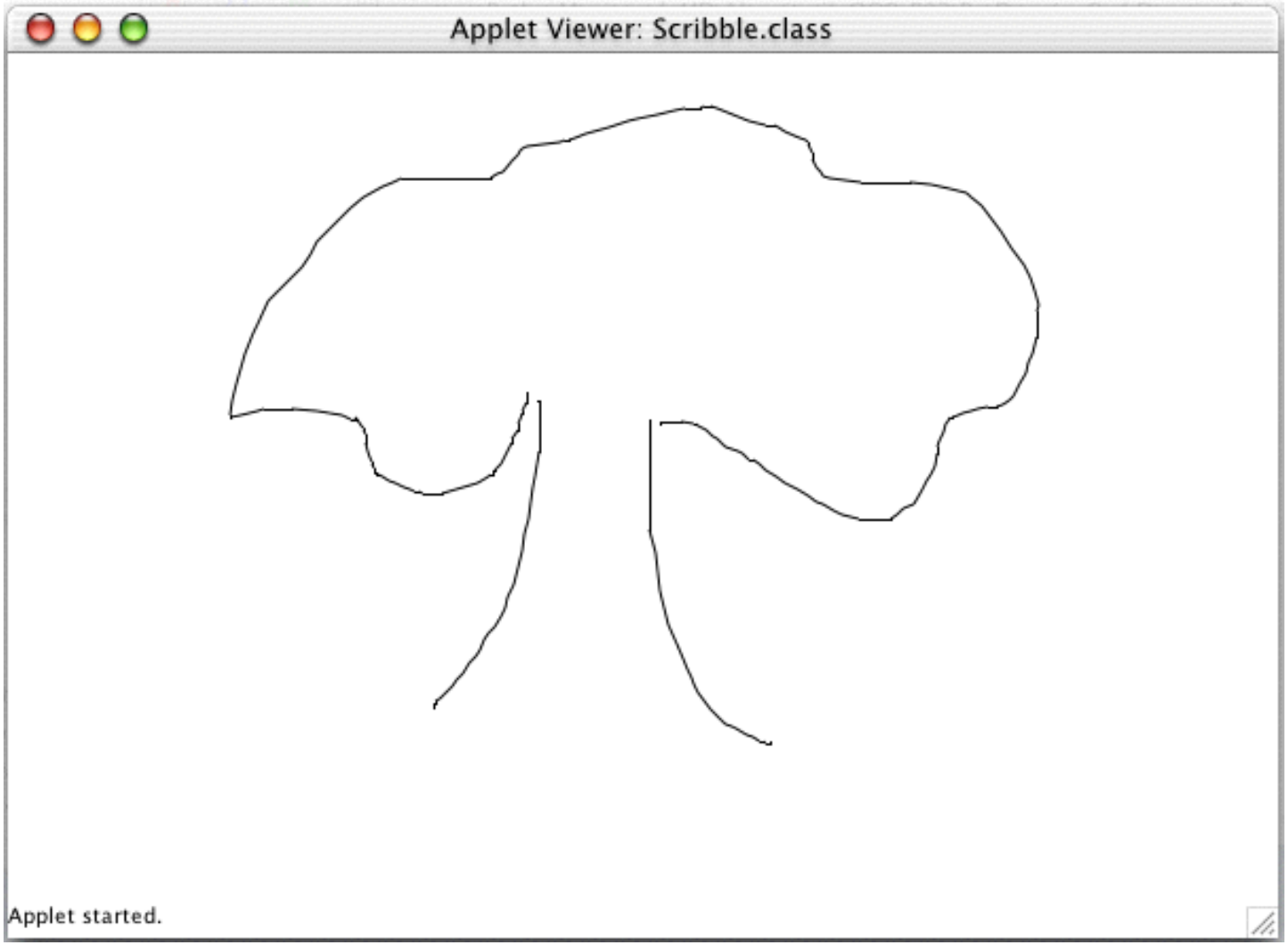
3. Faktorisering med henblik på udvidelse.

4. Muliggør tegning af linjestykker, rektangler og ovaler.

5. Muliggør tegning af udfyldte rektangler og ovaler.

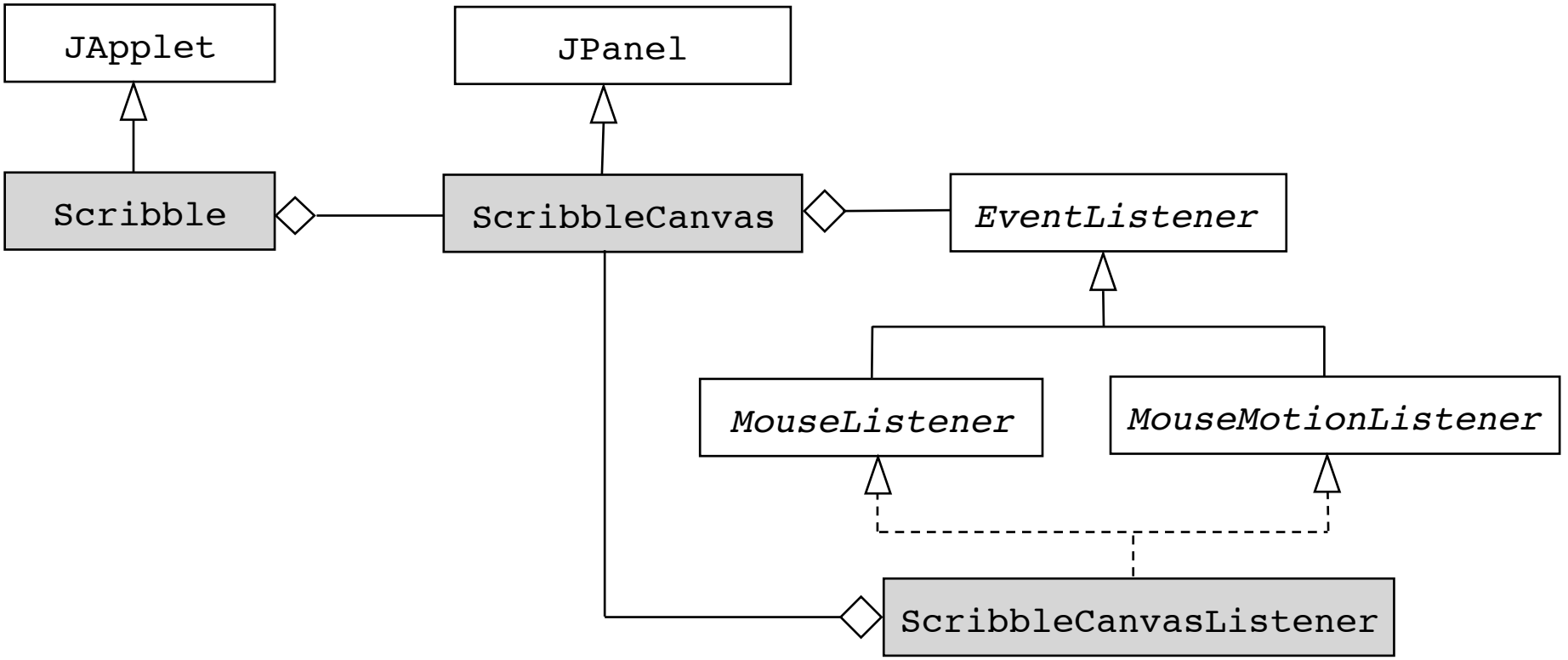
6. Muliggør tilføjelse af tekst via tastaturet.

Iteration 1



Programstruktur

(iteration 1)



Klassen `ScribbleCanvas`

```
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;
    protected int x, y;
}
```

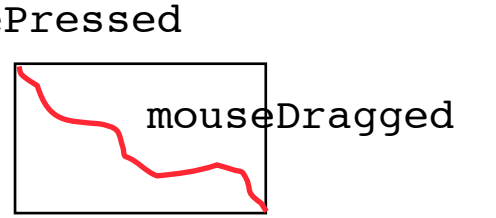
Klassen `ScribbleCanvasListener`

```
public class ScribbleCanvasListener
             implements MouseListener, MouseMotionListener {
    public ScribbleCanvasListener(ScribbleCanvas canvas) {
        this.canvas = canvas;
    }

    // ... mousePressed, mouseDragged, mouseReleased

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

    protected ScribbleCanvas canvas;
}
```

fortsættes

Alternativ: brug klassen `MouseInputAdapter` fra Swing

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

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

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



Brug af `mouseButtonDown` er overflødig

Klassen `Scribble`
(applet)

```
public class Scribble extends JApplet {
    public Scribble() {
        canvas = new ScribbleCanvas();
        canvas.setBackground(Color.white);
        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(canvas, BorderLayout.CENTER);
    }

    protected ScribbleCanvas canvas;
}
```

Klassen `Scribble`
(applet/application)

```
public class Scribble extends JApplet {
    public Scribble() {
        canvas = new ScribbleCanvas()
        canvas.setBackground(Color.white);
        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(canvas, BorderLayout.CENTER);
    }

    public static void main(String[] args) { ... }

    protected ScribbleCanvas canvas;
}
```

fortsættes

Metoden `main` i `Scribble`

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

Idiom til at gøre et program til både en applet og en applikation

```
public class DualAppletApp extends JApplet {
    protected boolean isApplet;

    public DualAppletApp() {
        this(true);
    }

    private DualAppletApp(boolean isApplet) {
        this.isApplet = isApplet;
        // ... general constructor stuff goes here
        if (!isApplet) {
            init();
            start();
        }
    }

    public static void main(String[] args) { ... }
}
```

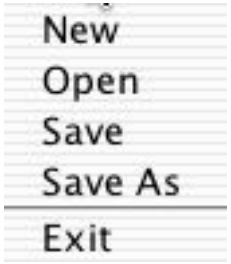
fortsættes

```
public static void main(String[] args) {
    JFrame frame = new JFrame();
    frame.setTitle(title);
    frame.getContentPane().setLayout(new BorderLayout());
    frame.getContentPane().add(
          new DualAppletApp(false), BorderLayout.CENTER);
    frame.addWindowListener(new FrameListener);
    frame.setSize(width, height);
    frame.pack();
    frame.setVisible(true);
}
```

FrameListener kan f.eks. være

```
WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
}
```

Iteration 2
(menuer, dialoger og filer)

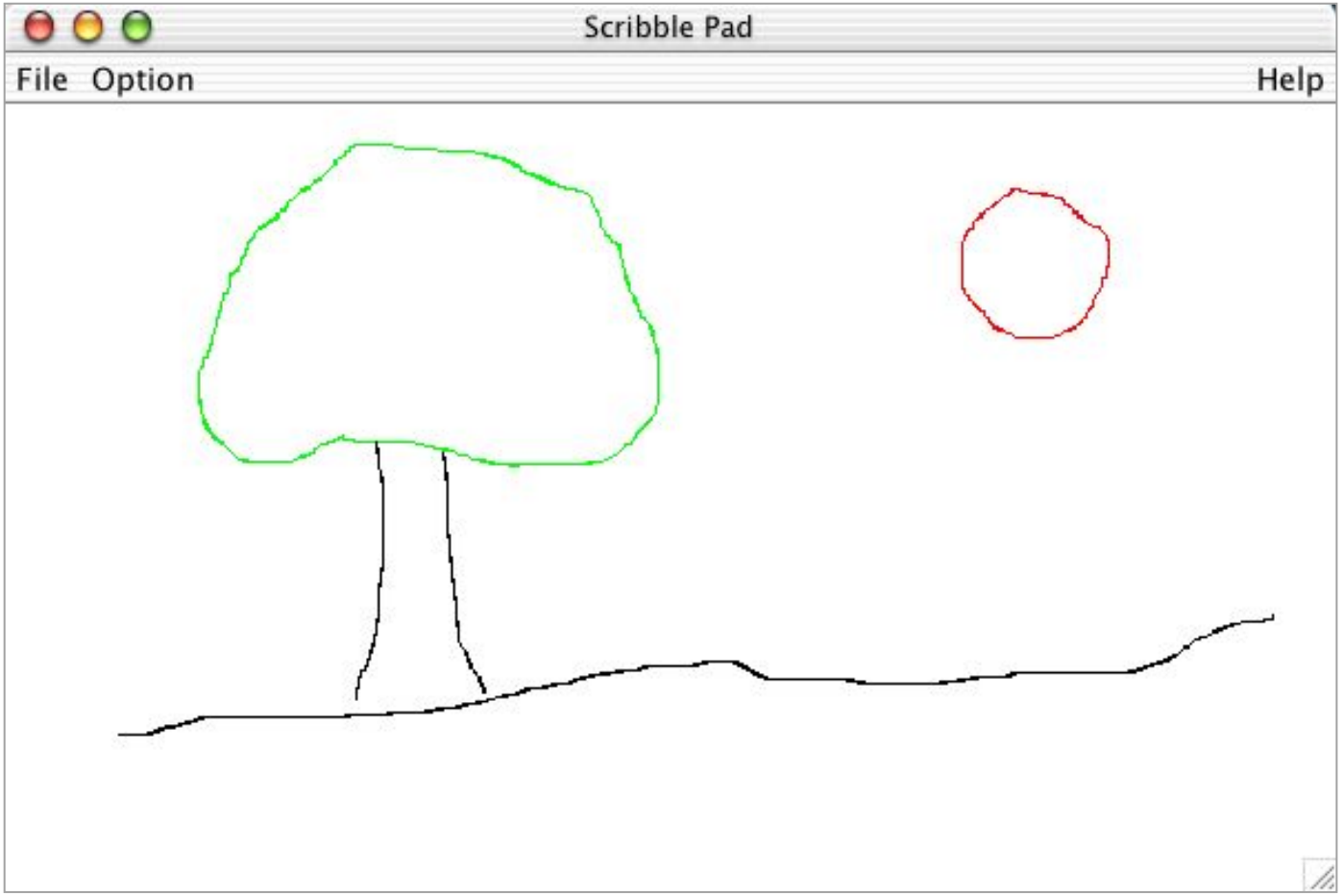


Tilføjer følgende funktionalitet:

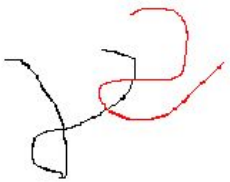
- Intern lagring af tegninger (så de kan gentegnes, f.eks. hvis vinduesstørrelsen ændres)
- Ekstern lagring af tegninger (i filer)
- Mulighed for valg af penfarve

Valg foretages ved hjælp af menuer og dialoger.

Iteration 2



Pennestrøg



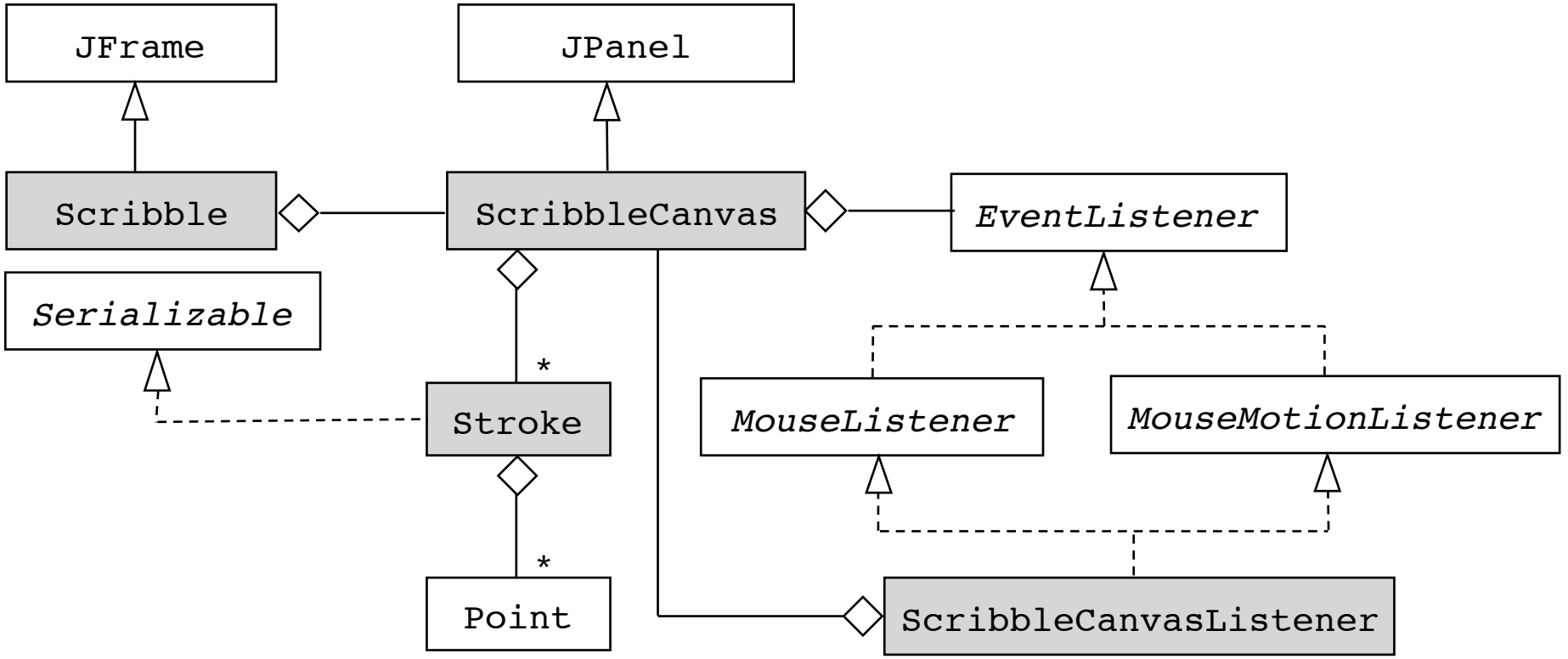
2 strøg

En tegning lagres i følgende struktur:

- Hver tegning består af en liste af pennestrøg (strokes)
- Hvert pennestrøg består af en liste af punkter og den anvendte penfarve

Programstruktur

(iteration 2)



Klassen `Stroke`

```
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 List<Point> getPoints() { return points; }
    public void addPoint(Point p) { points.add(p); }

    protected List<Point> points = new ArrayList<Point>();
    protected Color color = Color.black;
}
```

Klassen `ScribbleCanvas`

```
public class ScribbleCanvas extends JPanel {
    public ScribbleCanvas() { ... }

    public void setCurColor(Color curColor) { ... }
    public Color getCurColor(); { ... }
    public void startStroke(Point p) { ... }
    public void addPointToStroke(Point p) { ... }
    public void endStroke(Point p) { ... }
    public void paint(Graphics g) { ... }
    public void saveFile(String fileName) { ... }
    public void openFile(String fileName) { ... }
    public void newFile() { ... }

    protected EventListener listener;
    protected int x, y;

    protected List<Stroke> Strokes = new ArrayList<Stroke>();
    protected Stroke curStroke = null;
    protected Color curColor = Color.black;
}
```

```
public void setCurColor(Color curColor) {
    this.curColor = curColor;
}

public Color getCurColor() { return curColor; }

public void startStroke(Point p) {
    curStroke = new Stroke(curColor);
    curStroke.addPoint(p);
}

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

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

```
public void paint(Graphics g) {
    Dimension dim = getSize();
    g.setColor(Color.white);
    g.fillRect(0, 0, dim.width, dim.height);
    for (Stroke stroke : strokes) {
        g.setColor(stroke.getColor());
        Point prev = null;
        for (Point cur : stroke.getPoints()) {
            if (prev != null)
                g.drawLine(prev.x, prev.y, cur.x, cur.y);
            prev = cur;
        }
    }
}
```

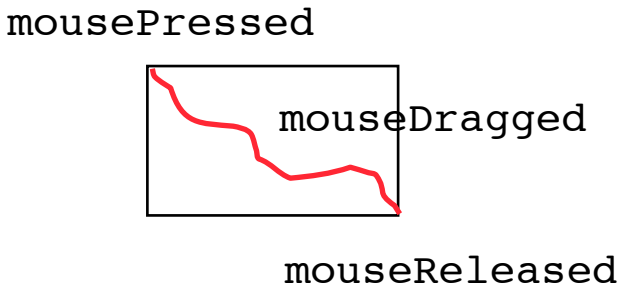
```
public void saveFile(String filename) {
    try {
        ObjectOutputStream out = new ObjectOutputStream(
                                        new FileOutputStream(filename));
        out.writeObject(strokes);
        out.close();
        System.out.println("Save drawing to " + filename);
  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

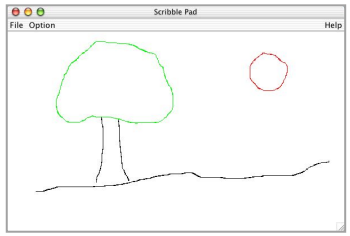
```
public void openFile(String filename) {
    try {
        ObjectInputStream in = new ObjectInputStream(
                                    new FileInputStream(filename));
        strokes = (List) in.readObject();
        in.close();
        repaint();
    } catch (IOException e1) {
        System.out.println("Unable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

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

Klassen `ScribbleCanvasListener`

```
public class ScribbleCanvasListener extends MouseInputAdapter {
    public ScribbleCanvasListener(ScribbleCanvas canvas) {
        this.canvas = canvas;
    }

    // mousePressed, mouseDragged, mouseReleased

    protected ScribbleCanvas canvas;
}
```

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

public void mouseDragged(MouseEvent e) {
    Point p = e.getPoint();
    canvas.getGraphics().
    drawLine(canvas.x, canvas.y, p.x, p.y);
    canvas.x = p.x;
    canvas.y = p.y;
    canvas.addPointToStroke(p);
}

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






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

    ... // methods (including main) and inner classes

    protected ScribbleCanvas canvas;
    protected String currentFileName;
    protected ActionListener exitAction;
    protected JFileChooser chooser = new JFileChooser(".");
}
```

Ændringer i applikationen

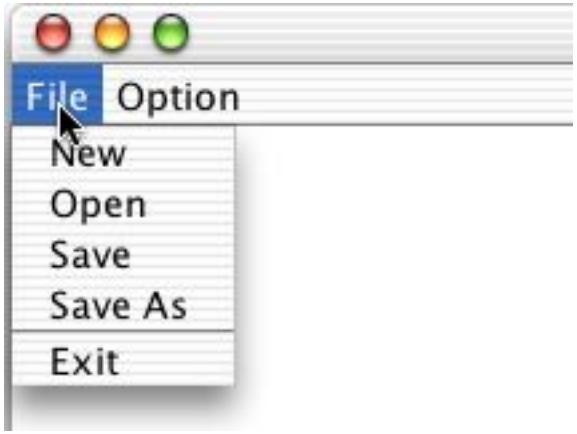
Nye metoder:

- `createMenuBar()`
- `newFile()`
- `openFile()`
- `saveFile()`
- `saveFileAs()`

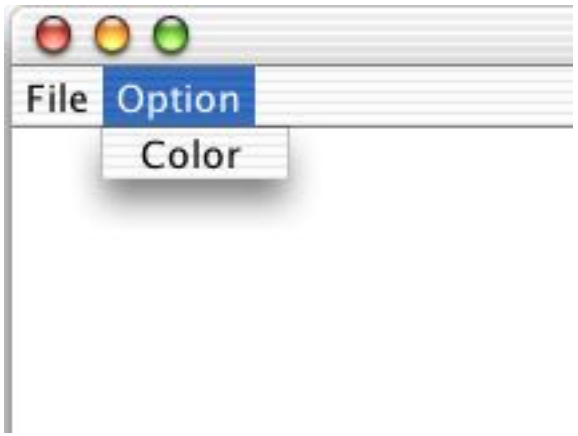
Indlejrede lytterklasser:

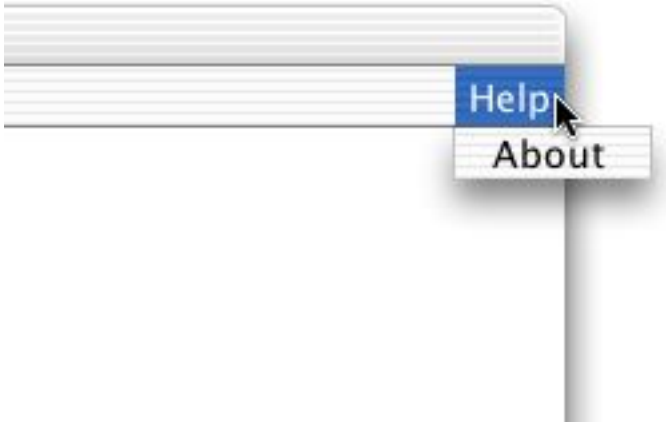
- `NewFileListener`
- `OpenFileListener`
- `SaveFileListener`
- `SaveAsFileListener`
- `ExitListener`
- `ColorListener`
- `AboutListener`

```
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());
```

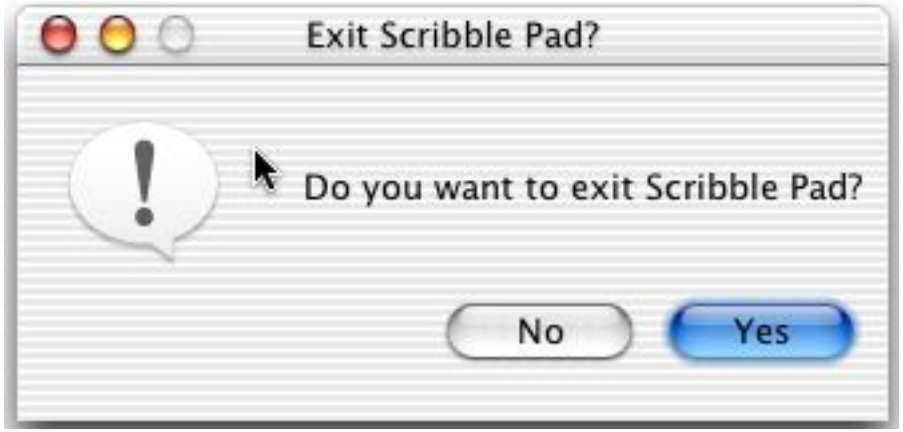


```
        exitAction = new ExitListener();
        mi = new JMenuItem("Exit");
        menu.add(mi);
        mi.addActionListener(exitAction);

        // 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;
    }
```



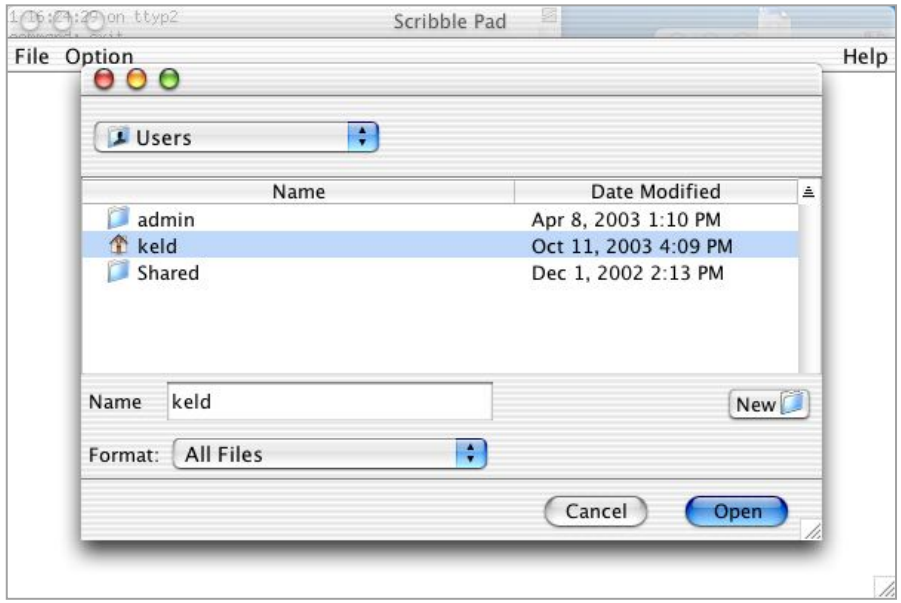


```
protected void newFile() {
    currentFilename = null;
    canvas.newFile();
    setTitle("Scribble Pad");
}

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 + "]");
}
```

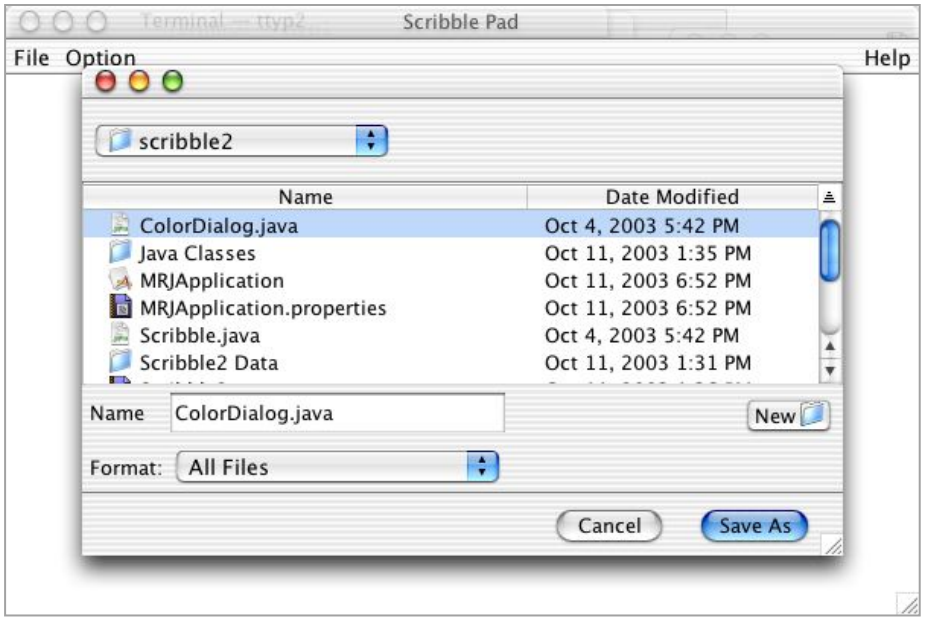



```
class ExitListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        int result = JOptionPane.showConfirmDialog(null,
            "Do you want to exit Scribble Pad?",
            "Exit Scribble Pad?",
            JOptionPane.YES_NO_OPTION);
        if (result ==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

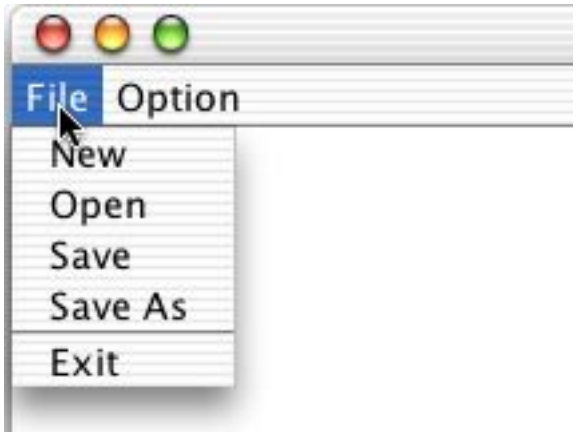

```
class AboutListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(null,
            "DrawingPad version 1.0\nCopyright (c) Xiaoping Jia 2002",
            "About",
            JOptionPane.INFORMATION_MESSAGE);
    }
}
```



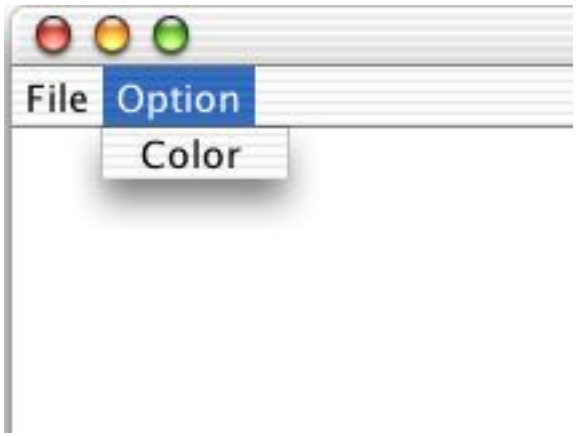

```
class OpenFileListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        int retval = chooser.showDialog(null, "Open");
        if (retval == JFileChooser.APPROVE_OPTION) {
            File theFile = chooser.getSelectedFile();
            if (theFile != null && theFile.isFile())
                String filename = theFile.getAbsolutePath();
                openFile(filename);
            }
        }
    }
}
```



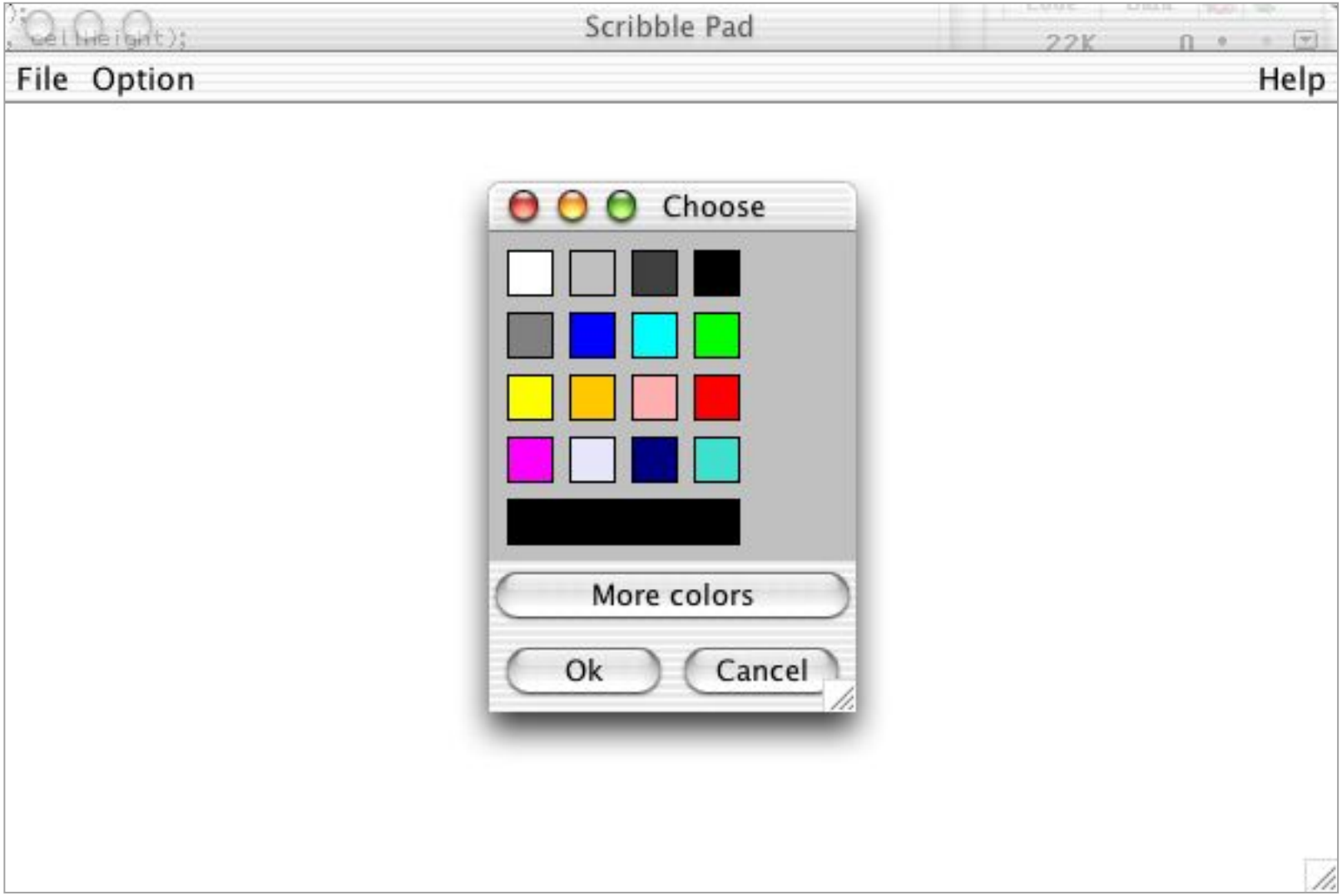

```
class SaveAsFileListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        int result = chooser.showDialog(null, "Save As");
        if (result == JFileChooser.APPROVE_OPTION) {
            File theFile = chooser.getSelectedFile();
            if (theFile != null && !theFile.isDirectory()) {
                String filename = theFile.getAbsolutePath();
                saveFileAs(filename);
            }
        }
    }
}
```




```
class NewFileListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        newFile();
    }
}


class SaveFileListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        saveFile();
    }
}
```

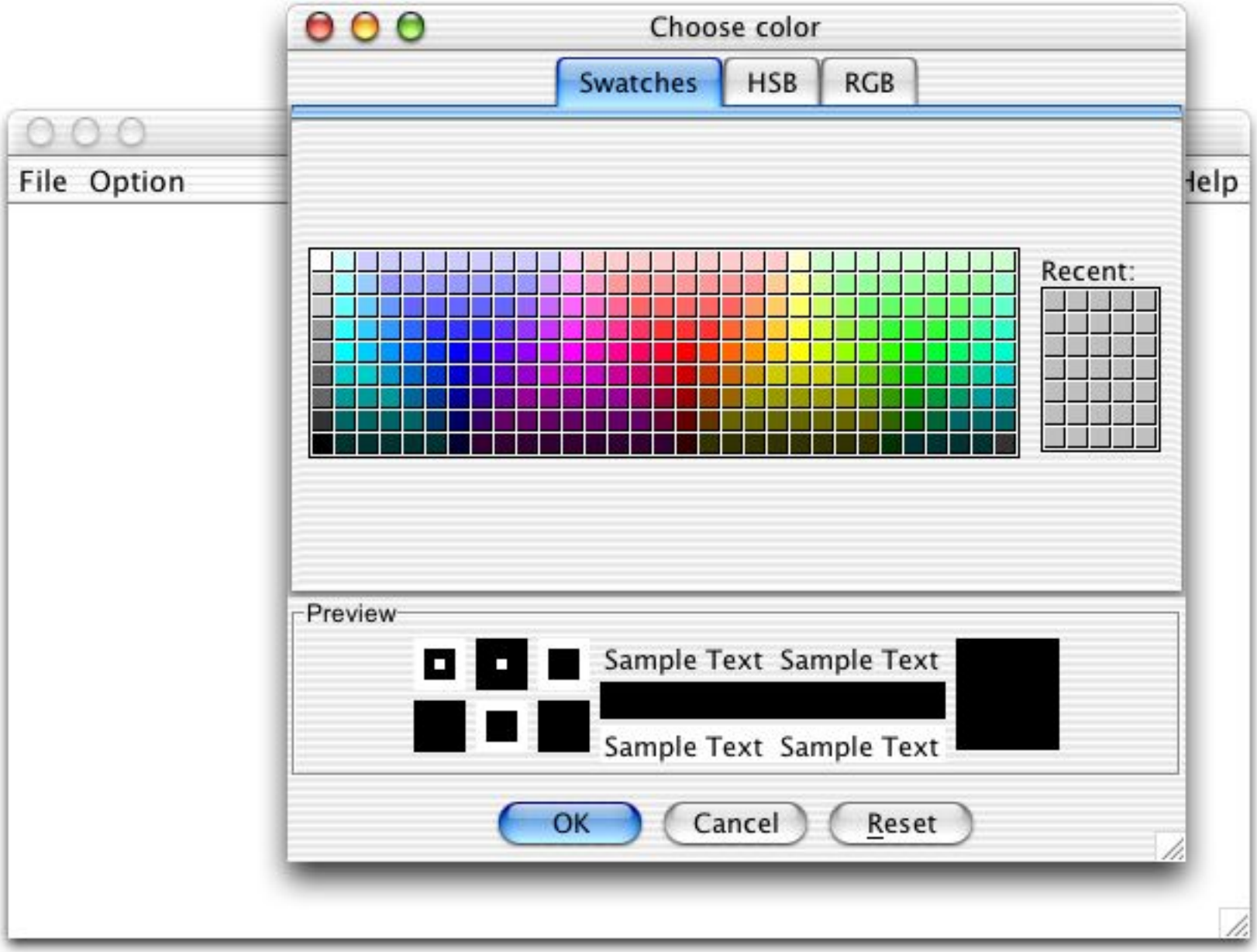



```
class ColorListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        Color result = dialog.showDialog();
        if (result != null)
            canvas.setCurColor(result);
    }

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


Scribble Pad
File Option
Help
Choose
More colors
Ok
Cancel

```
public class ColorDialog extends JDialog implements ActionListener {
    public ColorDialog(JFrame owner, String title, Color color) { ... }

    public Color showDialog() { ... }

    public void actionPerformed(ActionEvent e) { ... }

    protected JButton okButton;
    protected JButton cancelButton;
    protected JButton moreColorButton;

    protected ColorPanel colorPanel;
    protected JColorChooser chooser = new JColorChooser();
    protected Color color, result;

    class ColorPanel extends JPanel { ... }
}
```

Implementation: se bogen


Choose color
Swatches
HSB
RGB
File Option
Recent:
Preview
Sample Text Sample Text
Sample Text Sample Text
OK
Cancel
Reset

En simpel `ColorListener`

```
class ColorListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        Color result = JColorChooser.showDialog(
            Scribble.this, "Choose color", Color.BLACK);
        if (result != null)
            canvas.setCurColor(result);
    }
}
```

Iteration 3

faktorisering for at opnå udvidelighed

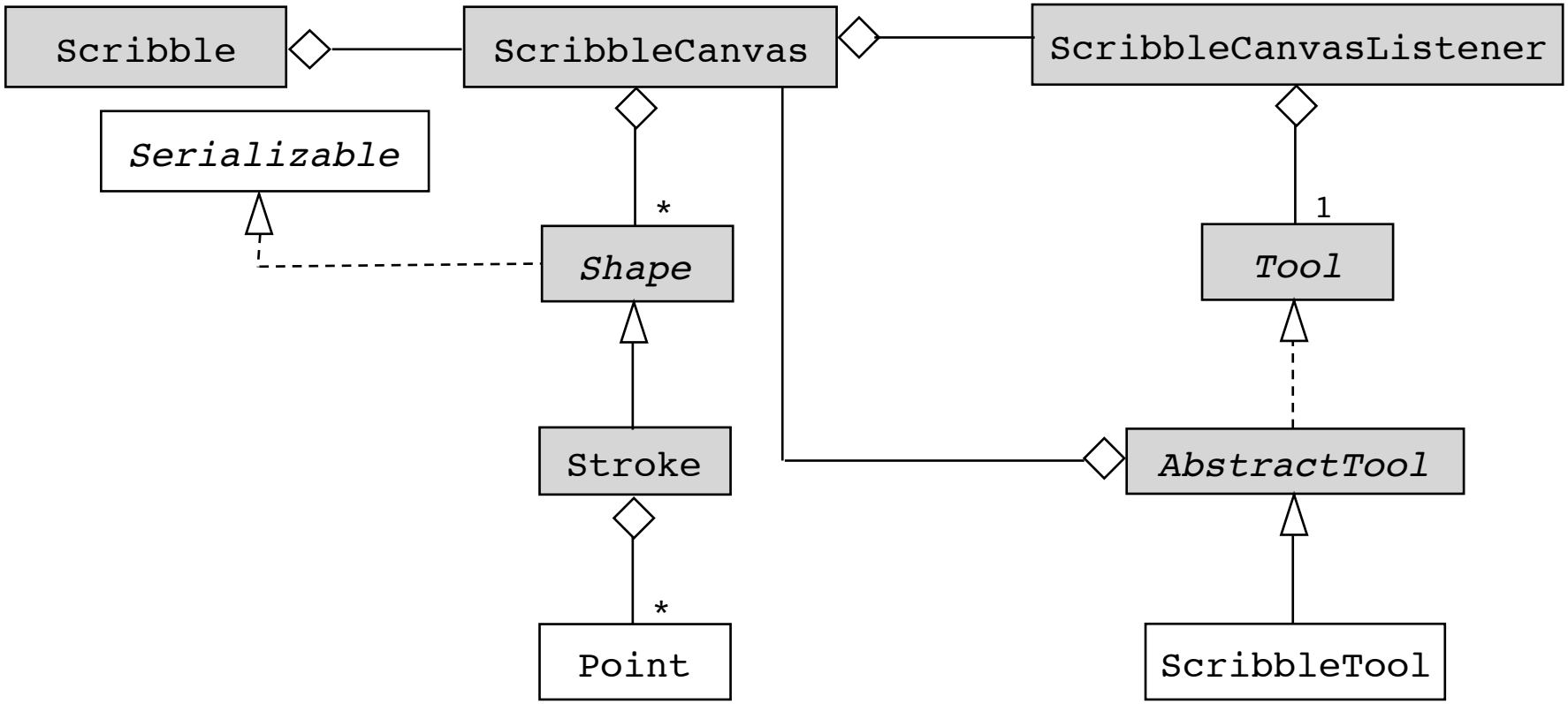
(1) Indførelse af en “form” (Shape), hvor et strøg er et særtilfælde.

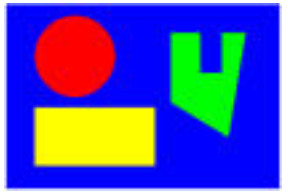
(2) Indførelse af et “værktøj” (Tool), der for hver mulig form reagerer på brugerhandlinger med musen.

Funktionaliteten er den samme som for iteration 2

Programstruktur

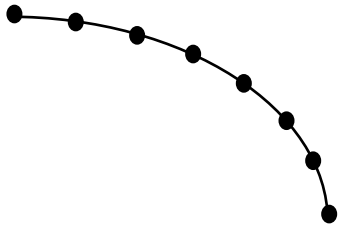
(iteration 3)





```
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;
}
```

Bemærk brugen af designmønsteret Strategy.
`Shape` er en abstrakt strategi, `Stroke` en konkret strategi.



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

    public List<Point> getPoints() { return points; }
    public void addPoint(Point p) { points.add(p); }

    public void draw(Graphics g) {
        if (color != null)
            g.setColor(color);
        Point prev = null;
        for (Point cur : points) {
            if (prev != null)
                g.drawLine(prev.x, prev.y, cur.x, cur.y);
            prev = cur;
        }
    }

    protected List<Point> points = new ArrayList<Point>();
    // color and associate methods moved to the superclass Shape
}
```

Hvorledes håndteres flere værktøjer?

En mulighed er at vedligeholde en heltalsvariabel, `currentTool`, der kan antage en af følgende værdier:

```
public static final int SCRIBBLE_TOOL  = 0;
public static final int LINE_TOOL      = 1;
public static final int RECTANGLE_TOOL = 2;
public static final int OVAL_TOOL      = 3;
```

Alternativt benytte Java 5.0's typesikre `enum`-konstruktion:

```
public enum ToolType { SCRIBBLE_TOOL, LINE_TOOL,
                       RECTANGLE_TOOL, OVAL_TOOL };
```

`java.sun.com/j2se/1.5.0/docs/guide/language/enums.html`

`mousePressed`, `mouseDragged` og `mouseReleased` i klassen `ScribbleCanvasListener` ændres, så de får følgende struktur:

```
Point p = e.getPoint();
switch (ScribbleCanvas.getCurrentTool()) {
case ScribbleCanvas.SCRIBBLE_TOOL:
    // ...
    break;
case ScribbleCanvas.LINE_TOOL:
    // ...
    break;
case ScribbleCanvas.RECTANGLE_TOOL:
    // ...
    break;
case ScribbleCanvas.OVAL_TOOL:
    // ...
    break;
}
```

En bedre løsning

Førnævnte løsning er ufleksibel.

En bedre løsning er at indkapsle adfærden for hvert værktøj i en særskilt klasse.

Som fælles grænseflade for værktøjerne defineres

```
public interface Tool {
    String getName();
    void startShape(Point p);
    void addPointToShape(Point p);
    void endShape(Point p);
}
```

```
public abstract class AbstractTool implements Tool {
    protected AbstractTool(ScribbleCanvas canvas, String name) {
        this.canvas = canvas;
        this.name = name;
    }

    public String getName() {
        return name;
    }

    protected ScribbleCanvas canvas;
    protected String name;
}
```

Bemærk brugen af designmønsteret Strategy.
`AbstractTool` er en abstrakt strategi, `ScribbleTool` en konkret strategi.

```
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;
}
```

Ændringer i `ScribbleCanvasListener`

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

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

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

protected Tool tool;
```

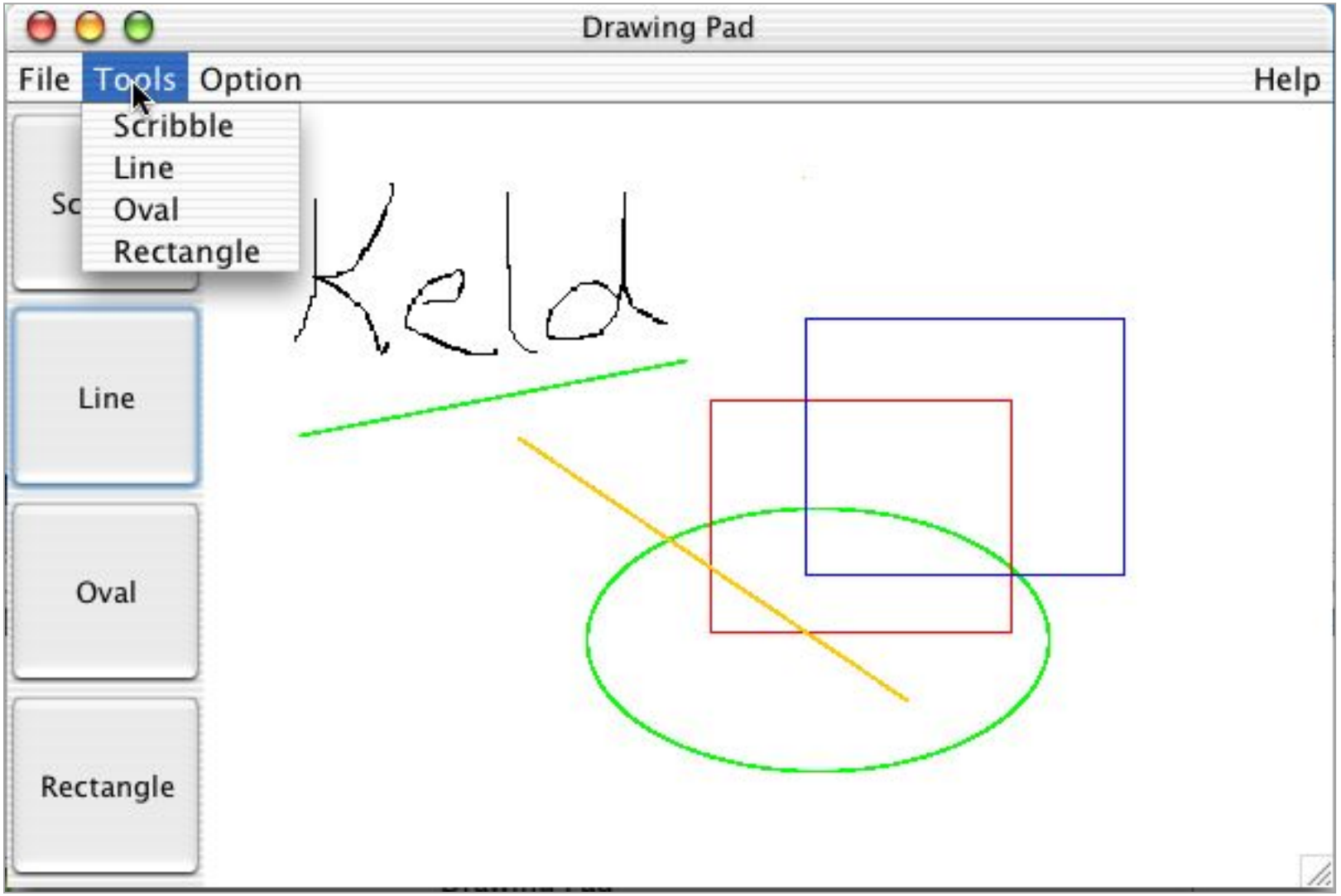
Ændringer i `Scribble`

```
public class Scribble extends JFrame {
   public Scribble(String title) {
       super(title);
       // calling the factory method
       canvas = makeCanvas();
       ...
     }
   }

   // the factory method
   protected ScribbleCanvas makeCanvas() {
       return new ScribbleCanvas();
   }

   ...
}
```

Ændringer i `ScribbleCanvas`

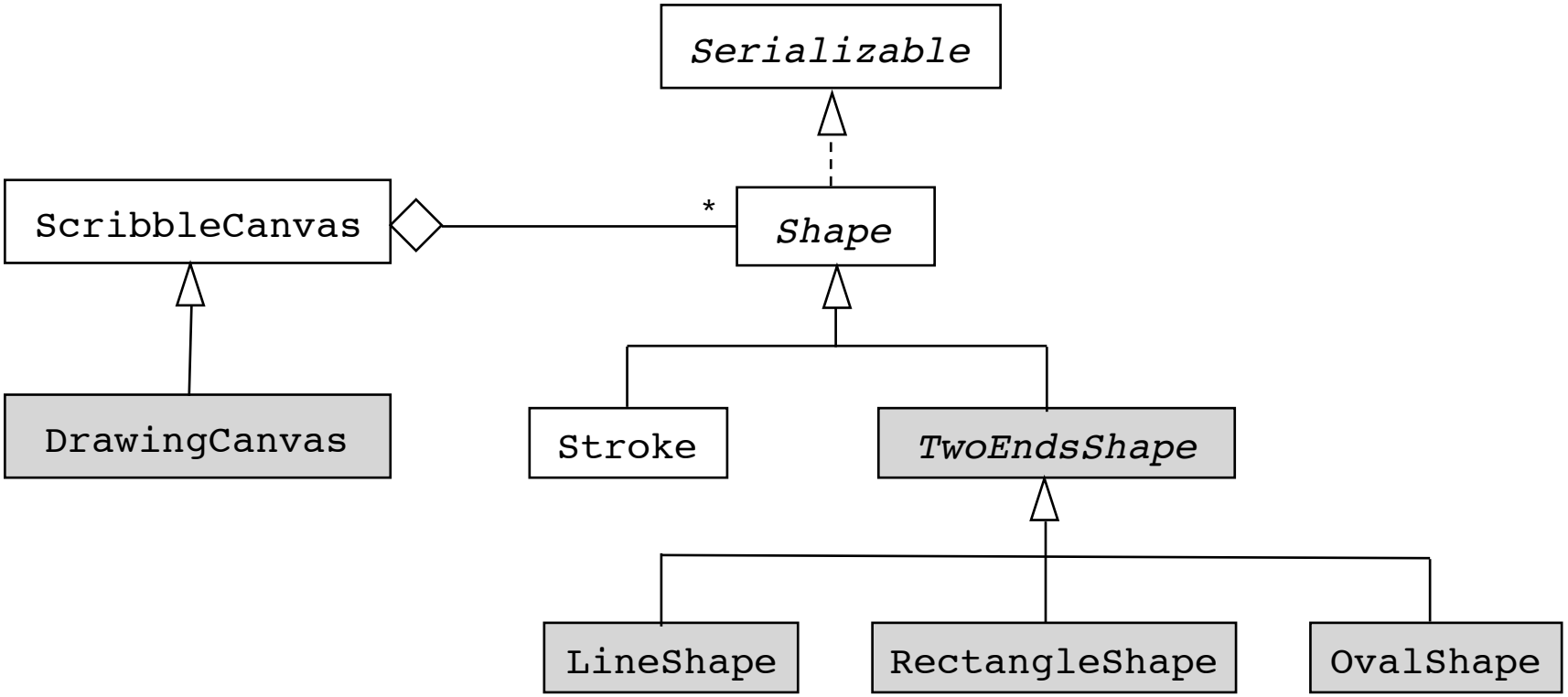

```
public class ScribbleCanvas extends JPanel {
    public ScribbleCanvas() {
        // calling factory method
        listener = makeCanvasListener();
        ....
    }

    // factory method
    protected EventListener makeCanvasListener() {
        return new ScribbleCanvasListener(this);
    }

    public void paint(Graphics g) {
        ...
        for (Shape shape : shapes)
            shape.draw(g);
    }

    List<Shape> shapes = new ArrayList<Shape>();
    ...
}
```

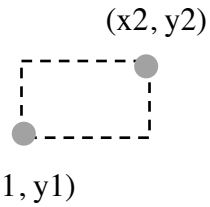
Iteration 4


Drawing Pad
File
Tools
Option
Help
Scribble
Line
Oval
Rectangle
Line
Oval
Rectangle

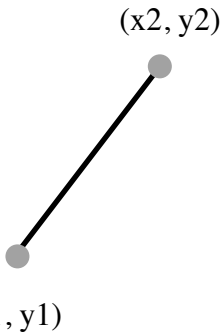
Programstruktur

(iteration 4, formerne)

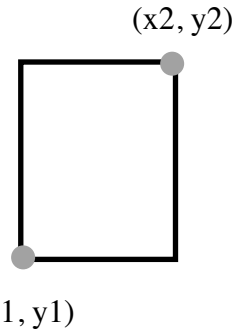




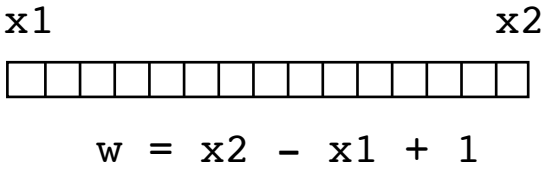

```
public abstract class TwoEndsShape extends Shape
                                    implements Cloneable {
    public TwoEndsShape() {}
    public TwoEndsShape(Color color) { super(color); }

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

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

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

    protected int x1, y1, x2, y2;
}
```

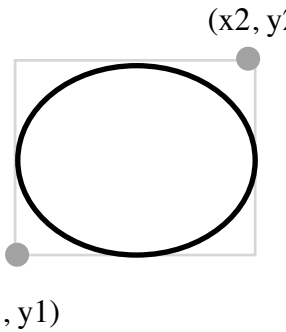



```
public class LineShape extends TwoEndsShape {
    public void draw(Graphics g) {
        g.setColor(color);
        g.drawLine(x1, y1, x2, y2);
    }

    public void drawOutline(Graphics g,
                            int x1, int y1, int x2, int y2) {
        g.drawLine(x1, y1, x2, y2);
    }
}
```



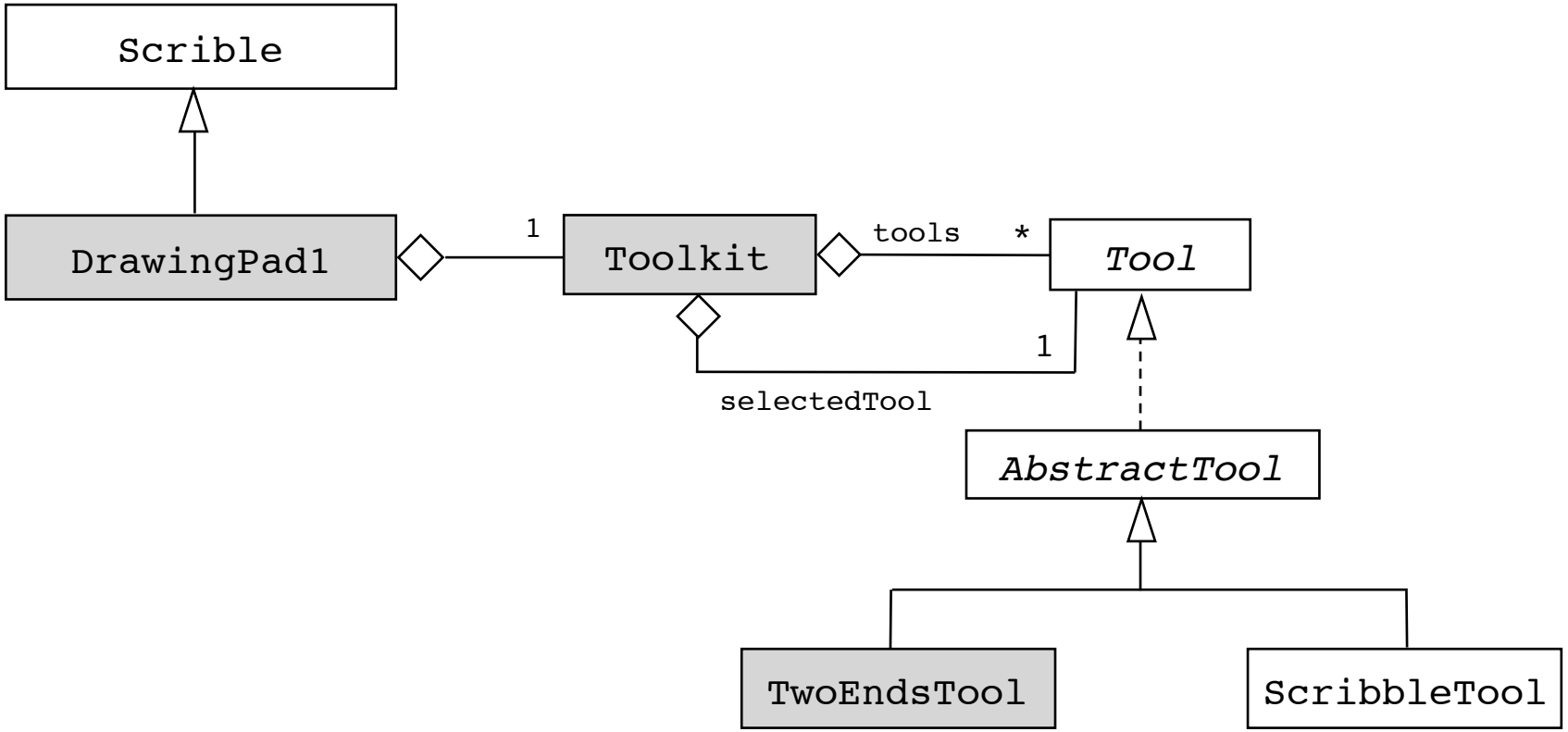

```
public class RectangleShape extends TwoEndsShape {
    public void draw(Graphics g) {
        int x = Math.min(x1, x2);
        int y = Math.min(y1, y2);
        int w = Math.abs(x2 - x1) + 1;
        int h = Math.abs(y2 - y1) + 1;
        g.setColor(color);
        g.drawRect(x, y, w, h);
    }

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






```
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;
        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);
    }
}
```

Programstruktur
(iteration 4, værktøjerne)




```
public class ToolKit {
   protected List<Tool> tools = new ArrayList<Tool>();
   protected Tool selectedTool;

   public void addTool(Tool tool) { tools.add(tool); }

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

   public Tool getTool(int i) { return tools.get(i); }

   public Tool getSelectedTool() { return selectedTool; }

   public void setSelectedTool(Tool tool) { selectedTool = tool; }

   public void setSelectedTool(int i) {
       Tool tool = getTool(i);
       if (tool != null)
           selectedTool = tool;
   }
```

fortsættes

```
public Tool findTool(String name) {
    for (Tool tool : tools)
        if (tool.getName().equals(name)))
            return tool;
    return null;
}
```

```
public Tool setSelectedTool(String name) {
    Tool tool = findTool(name);
    if (tool != null)
        selectedTool = tool;
    return tool;
}
```

Designmønsteret State

Kategori:

Adfærdsmæssigt designmønster

Hensigt:

At gøre det muligt, at et objekt kan ændre adfærd, når dets interne tilstand ændres.

Anvendelse:

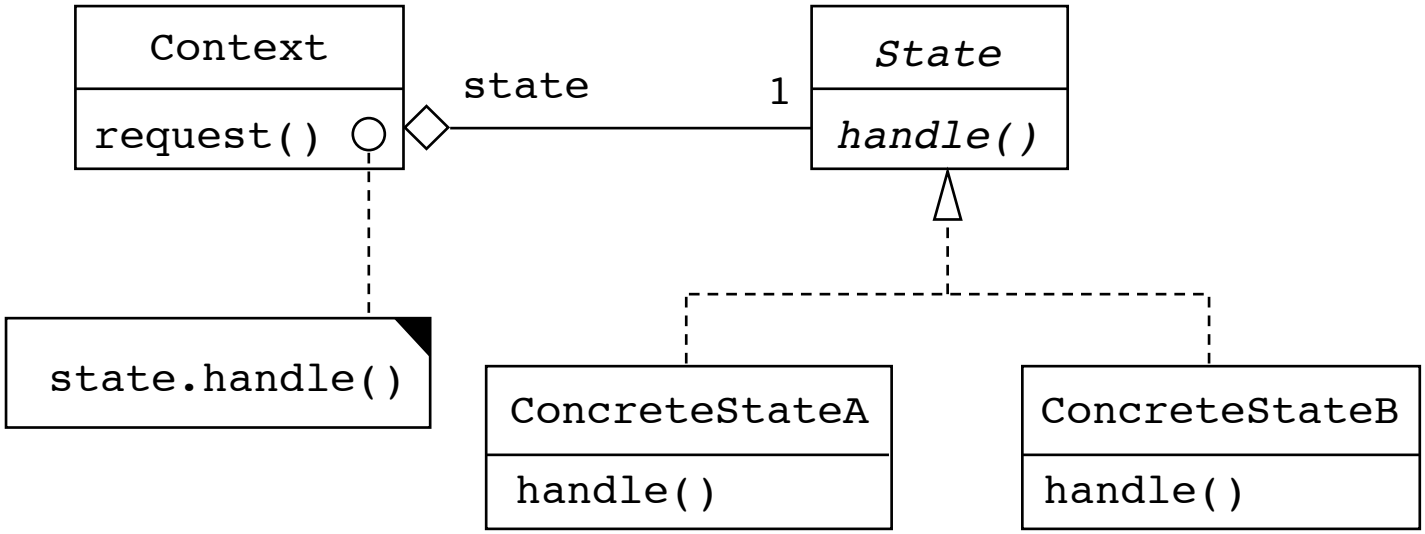
- Når et objekts adfærd afhænger at dets tilstand, og det må ændre adfærd på kørselstidspunktet, når dets tilstand ændres (f.eks. når et nyt værktøj vælges)
- Metoder har betingede flervejssætninger, der afhænger af objektets tilstand (f.eks. `switch`-sætninger i `ScribbleCanvasListener`).

Andre navne:

Objects for states

Designmønsteret State
(fortsat)

Struktur:



Designmønsteret State
(fortsat)

Deltagere:

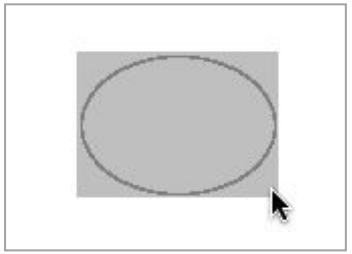
Context (f.eks. `Toolkit`), der har en instans af *ConcreteState*, der definerer den aktuelle tilstand (f.eks. `selectedTool`-feltet i `Toolkit`)

State (f.eks. `Tool`), der definerer en grænseflade til indkapsling af adfærden knyttet til en given tilstand for *Context*

ConcreteState (f.eks. `ScribbleTool`), hvor hver underklasse implementerer en adfærd (f.eks. `startShape`)

```
public class TwoEndsTool extends AbstractTool {
    public static final int LINE = 0, OVAL = 1, RECT = 2;

    public TwoEndsTool(ScribbleCanvas canvas, String name, int shape) {
        super(canvas, name);
        this.shape = shape;
    }

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

    protected int shape = LINE;
    protected int xStart, yStart;

    // helper methods
    private static void drawLine(Graphics g,
                                 int x1, int y1, int x2, int y2) { ... }
    private static void drawRect(Graphics g,
                                 int x, int y, int w, int h) { ... }
    private static void drawOval(Graphics g,
                                 int x, int y, int w, int h) { ... }
}
```

```
public void startShape(Point p) {
    xStart = canvas.x = p.x;
    yStart = canvas.y = p.y;
    Graphics g = canvas.getGraphics();
    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);
    }
}
```



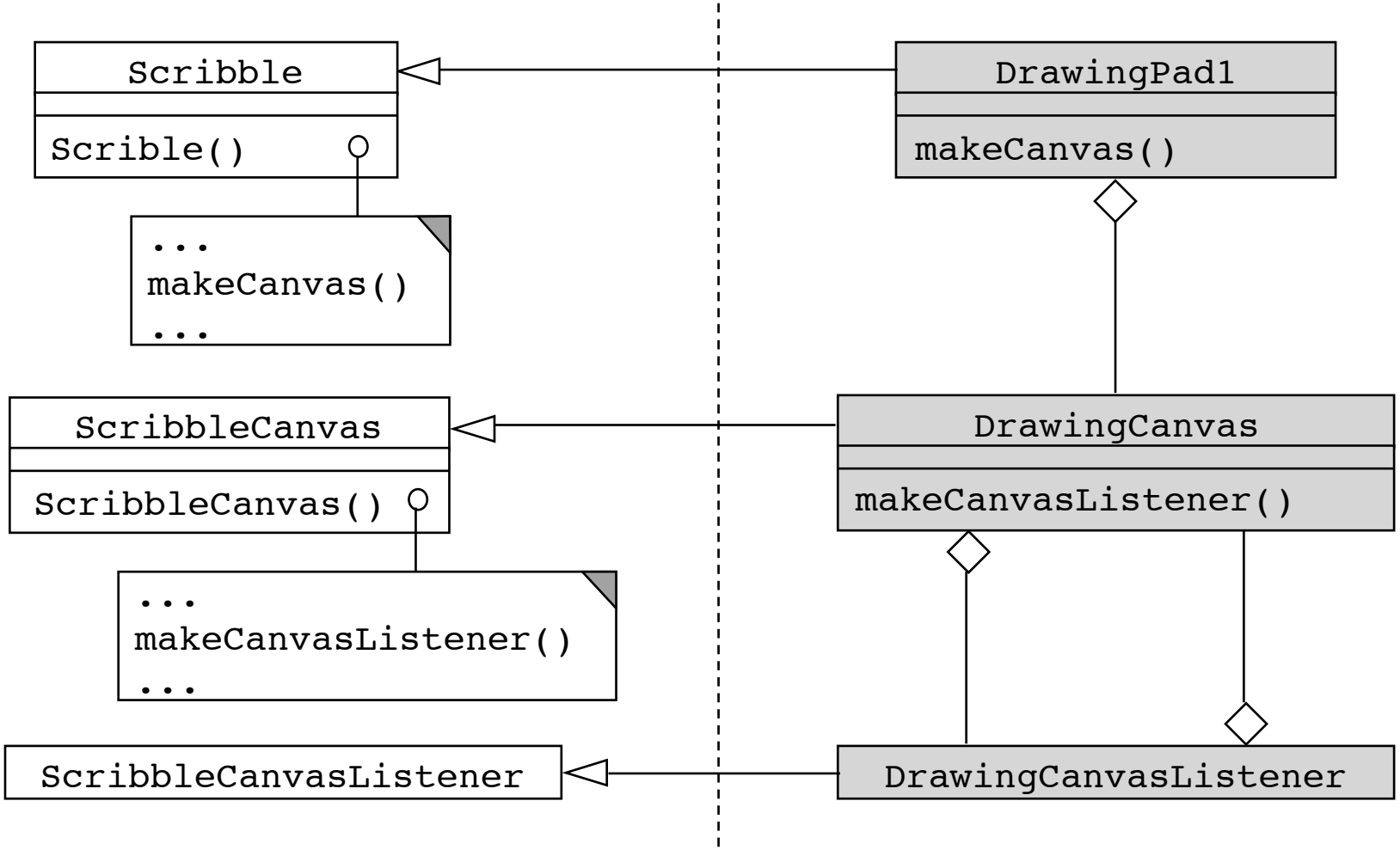
```
public void addPointToShape(Point p) {
    Graphics g = canvas.getGraphics();
    g.setXORMode(Color.white);
    g.setColor(Color.lightGray);
    switch (shape) {
    case LINE:
        drawLine(g, xStart, yStart, canvas.x, canvas.y);
        drawLine(g, xStart, yStart, p.x, p.y);
        break;
    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);
    }
    canvas.x = p.x;
    canvas.y = p.y;
}
```

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


```
private static void drawLine(Graphics g,
                             int x1, int y1, int x2, int y2) {
    g.drawLine(x1, y1, x2, y2);
}

private 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);
}

private 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);
}
```

Programstruktur

(iteration 4, overordnet)



Designmønsteret Factory Method

Kategori:

Konstruerende designmønster

Hensigt:

At definere en grænseflade til skabelse af et objekt, men lade underklasserne bestemme fra hvilken klasse, det skal skabes

Anvendelse:

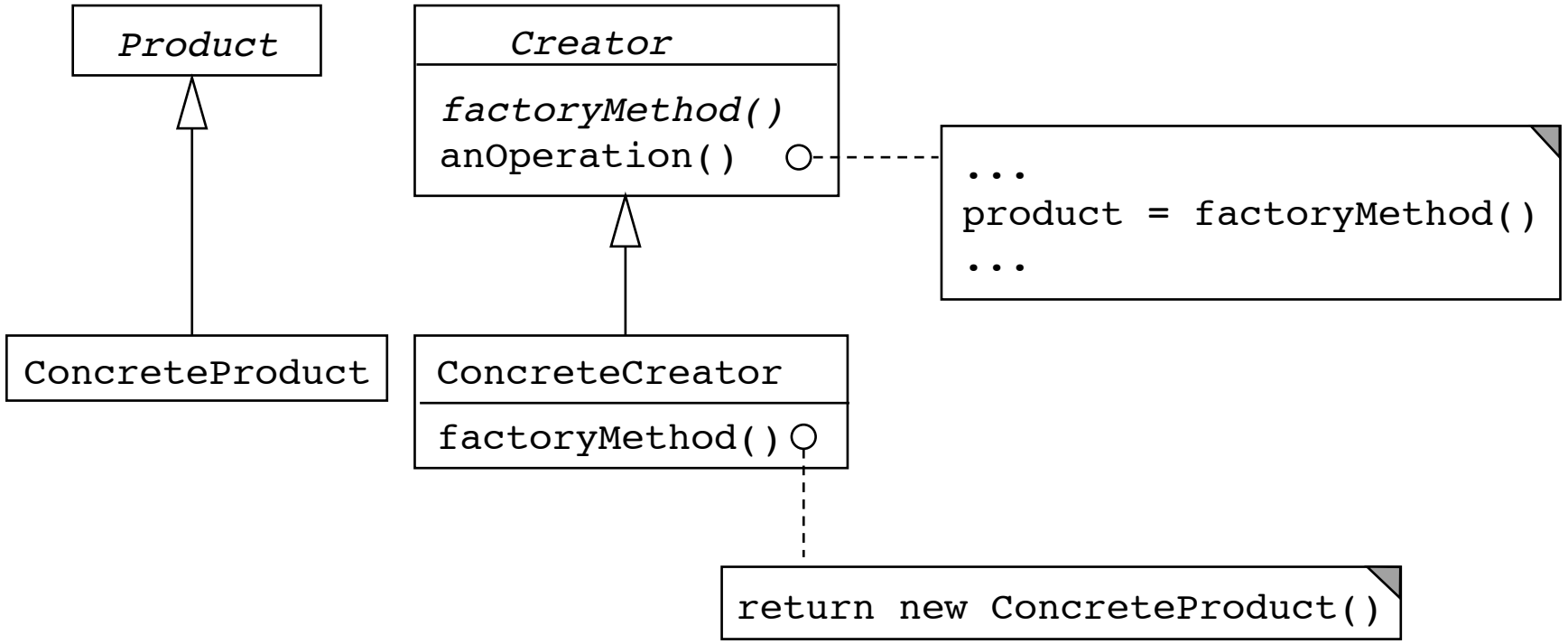
- Når en klasse ikke kan forudse, hvilken type objekt, den skal skabe
- Når en klasse ønsker, at dens underklasser skal afgøre, hvilke objekter, der skal skabes

Andre navne:

Virtual constructor

Designmønsteret Factory Method
(fortsat)

Struktur:



Designmønsteret Factory Method
(fortsat)

Deltagere:

Product (f.eks. `EventListener`), der definerer en grænseflade for de objekter, factory-metoden skaber

ConcreteProduct (f.eks. `DrawingCanvasListener`), der implementerer *Product*-grænsefladen

Creator (f.eks. `Scribble`), som erklærer en eller flere factory-metoder (f.eks. `makeCanvasListener`), som returnerer et objekt af typen *Product*.

ConcreteCreator (f.eks. `DrawingPad1`), der implementerer factory-metoden, så den returnerer et objekt af typen *ConcreteProduct*.

Klassen `DrawingCanvasListener` tillader brug af flere værktøjer

```
public class DrawingCanvasListener extends ScribbleCanvasListener {
    public DrawingCanvasListener(DrawingCanvas canvas) {
        super(canvas);
    }

    public Tool getTool() { return tool; }

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

Klassen `DrawingCanvas` tillader brug af flere værktøjer

```
public class DrawingCanvas extends ScribbleCanvas {
    public void setTool(Tool tool) {
        drawingCanvasListener.setTool(tool);
    }

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

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

    protected DrawingCanvasListener drawingCanvasListener;
}
```

```
public class DrawingPad1 extends Scribble {
    public DrawingPad1(String title) {
        super(title);
        initTools();
        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);
                }
            }
        };
        JComponent toolbar = createToolBar(toolListener);
        getContentPane().add(toolbar, BorderLayout.WEST);
        JMenu menu = createToolMenu(toolListener);
        menuBar.add(menu, 1); // insert at index position 1
    }
```

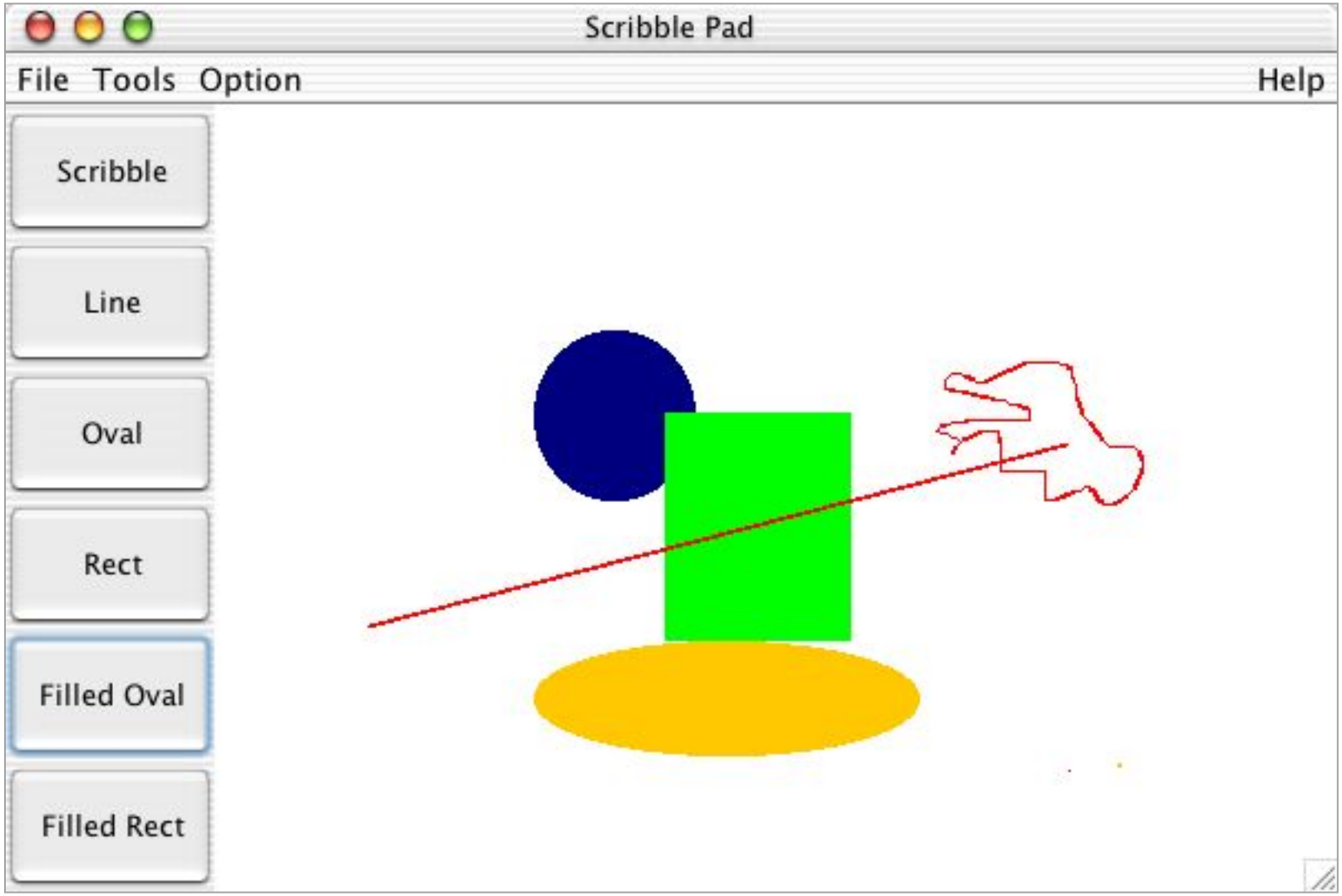
fortsættes

```
protected ToolKit toolkit;
protected DrawingCanvas drawingCanvas;

public Tool getSelectedTool() {
    return toolkit.getSelectedTool();
}

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));
}

protected ScribbleCanvas makeCanvas() {
    return (drawingCanvas = new DrawingCanvas());
}
```

fortsættes

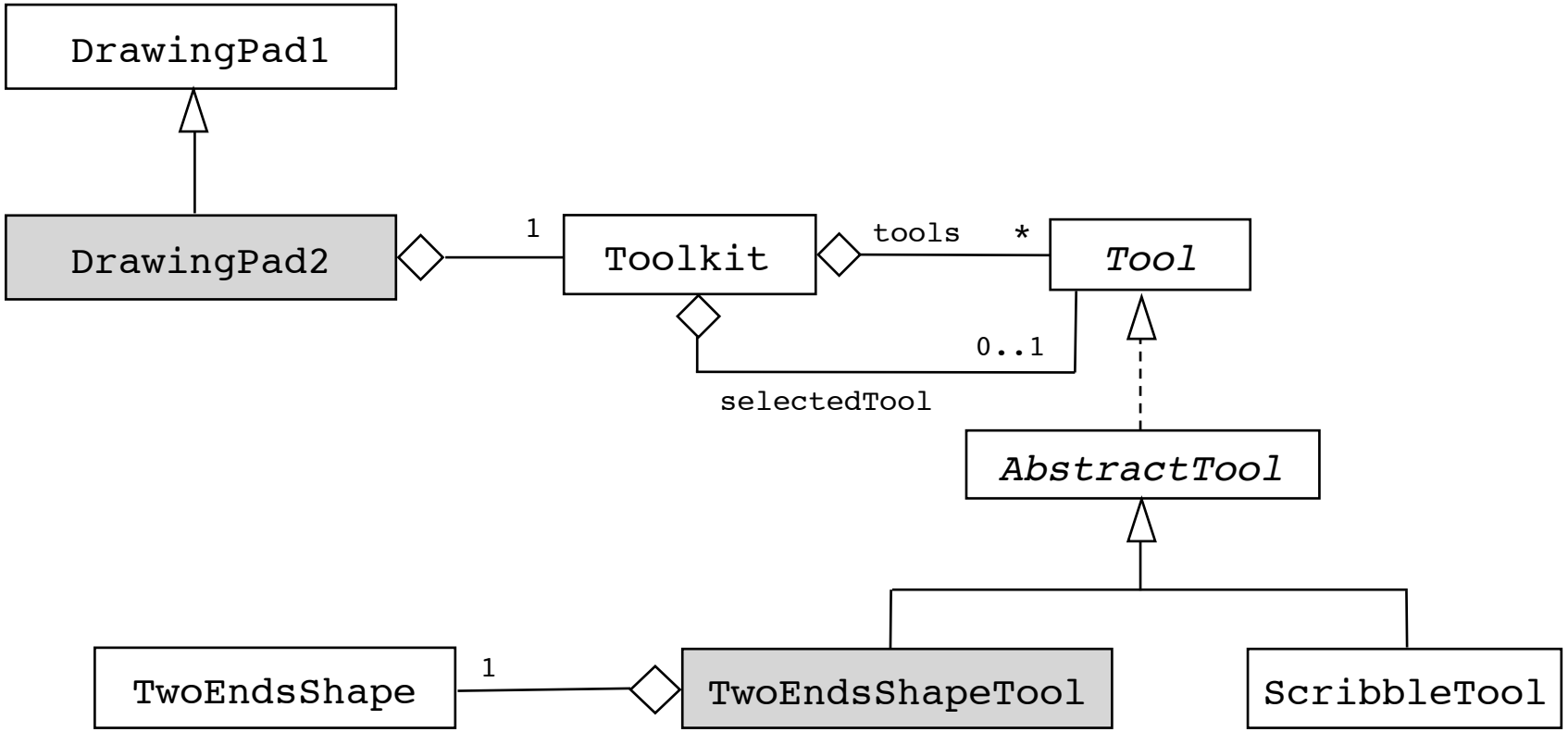
```
protected JComponent createToolBar(ActionListener toolListener) {
    JPanel toolbar = new JPanel(new GridLayout(0, 1));
    int n = toolkit.getToolCount();
    for (int i = 0; i < n; i++) {
        Tool tool = toolkit.getTool(i);
        JButton button = new JButton(tool.getName());
        button.addActionListener(toolListener);
        toolbar.add(button);
    }
    return toolbar;
}

protected JMenu createToolMenu(ActionListener toolListener) {
    JMenu menu = new JMenu("Tools");
    int n = toolkit.getToolCount();
    for (int i = 0; i < n; i++) {
        Tool tool = toolkit.getTool(i);
        JMenuItem menuitem = new JMenuItem(tool.getName());
        menuitem.addActionListener(toolListener);
        menu.add(menuitem);
    }
    return menu;
}
```

Iteration 5


Scribble Pad
File Tools Option
Help
Scribble
Line
Oval
Rect
Filled Oval
Filled Rect

Programstruktur

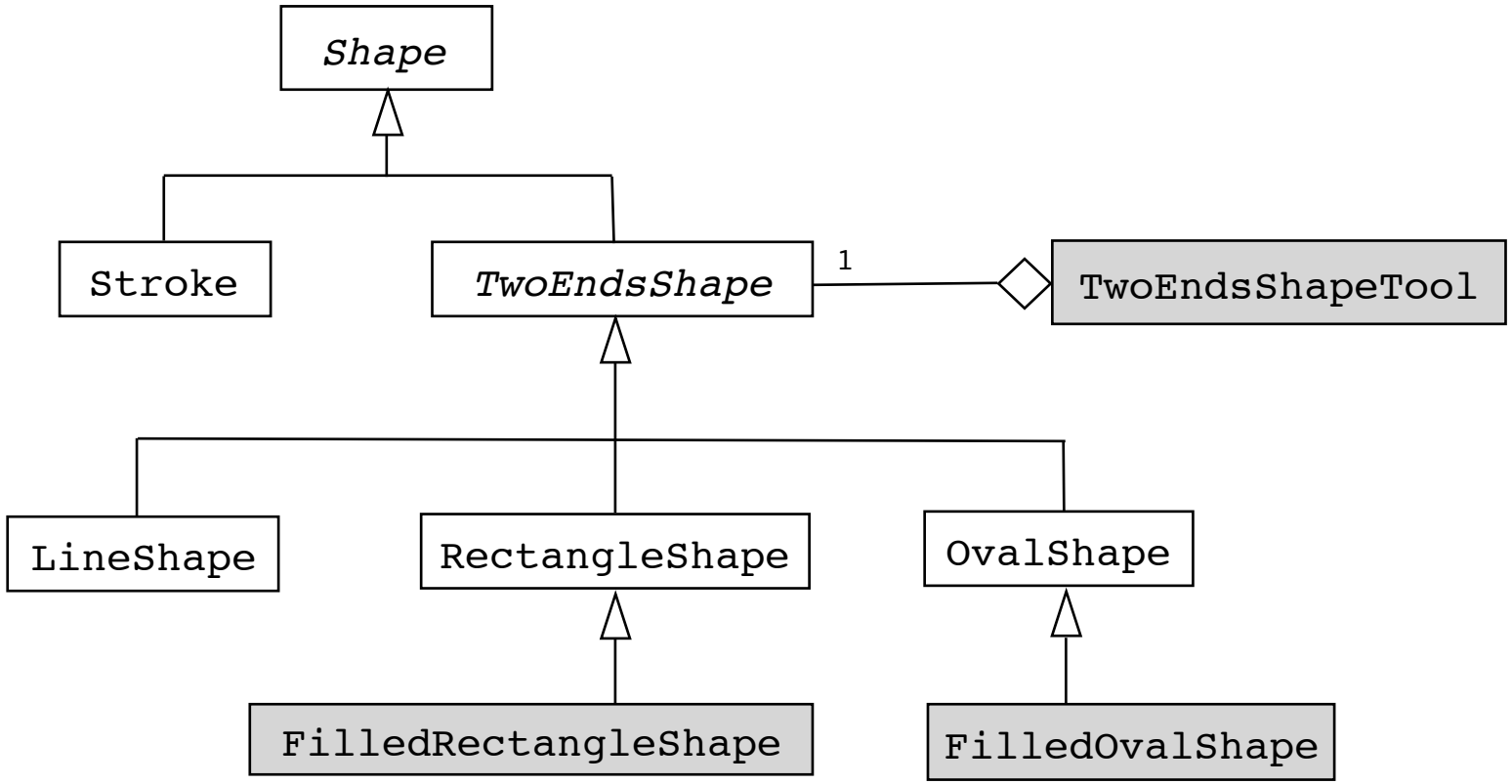
(iteration 5, værktøjer)

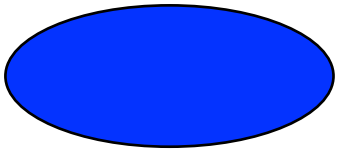


Ny klasse. Erstatter `TwoEndsTool`

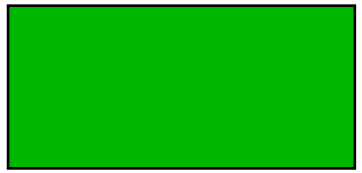
Programstruktur

(iteration 5, former)





```
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;
        g.setColor(color);
        g.fillOval(x, y, w, h);
    }
}
```



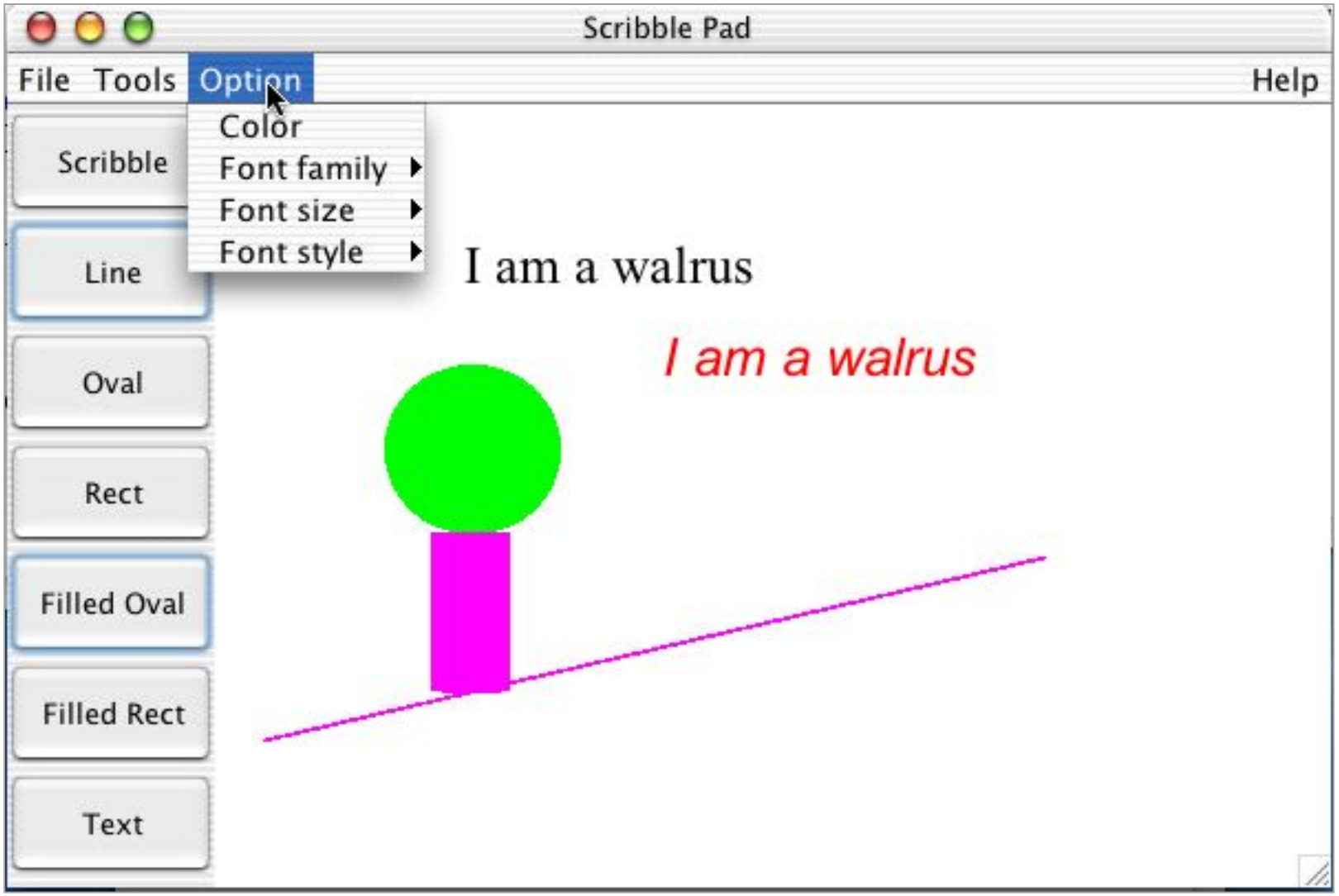
```
public class FilledRectangleShape extends RectangleShape {
    public void draw(Graphics g) {
        int x = Math.min(x1, x2);
        int y = Math.min(y1, y2);
        int w = Math.abs(x1 - x2) + 1;
        int h = Math.abs(y1 - y2) + 1;
        g.setColor(color);
        g.fillRect(x, y, w, h);
    }
}
```

```
public class TwoEndsShapeTool extends AbstractTool {
    public TwoEndsShapeTool(ScribbleCanvas canvas, String name,
                            TwoEndsShape prototype) {
        super(canvas, name);
        this.prototype = prototype;
    }

    protected int xStart, yStart;
    protected TwoEndsShape prototype;

    public void startShape(Point p) {
        xStart = canvas.x = p.x;
        yStart = canvas.y = p.y;
        Graphics g = canvas.getGraphics();
        g.setXORMode(Color.white);
        g.setColor(Color.lightGray);
        prototype.drawOutline(g, xStart, yStart, xStart, yStart);
    }
```

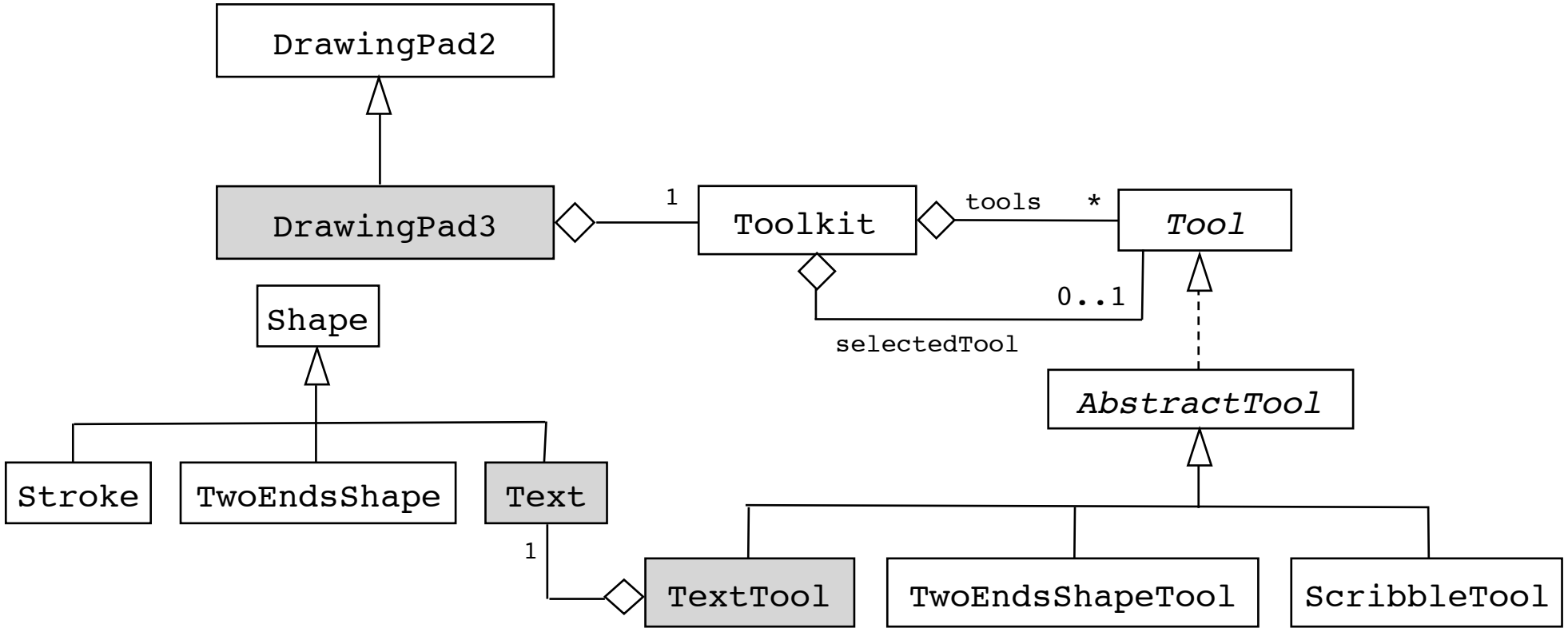
fortsættes

```
public void addPointToShape(Point p) {
    Graphics g = canvas.getGraphics();
    g.setXORMode(Color.white);
    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) {
    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();
}
```

Iteration 6


Scribble Pad
File
Tools
Option
Help
Color
Font family
Font size
Font style
Scribble
Line
Oval
Rect
Filled Oval
Filled Rect
Text
I am a walrus
I am a walrus

Programstruktur
(iteration 6)



Input fra tastatur

1. Kun én komponent ad gangen kan modtage input fra tastaturet.

3. For at modtage input fra tastaturet, må en komponent have **tastaturfokus**.

4. En komponent kan opnå tastaturfokus ved hjælp af metoden `requestFocus()`.

I `DrawingPad3` kaldes `canvas.requestFocus()`, når brugeren efter valg af `TextTool` klikker med musen på tegneområdet (`canvas`)

Valg af `Tool`

Et værktøj kan vælges såvel ved tryk på en knap på værktøjslinjen som ved valg af et menuemne (som også er en knap).

I begge tilfælde er aktionen den samme.

Grænsefladen `Action` er en udvidelse af `ActionListener`, som med fordel kan benyttes i tilfælde, hvor funktionaliteten er den samme for flere kontroller.

Konkrete klasser er ofte underklasser af den abstrakte klasse `AbstractAction`, som implementerer alle metoder i `Action`, med undtagelse af `actionPerformed`.

Java 2D

Tilbyder avancerede faciliteter til grafiske manipulationer.

- Klassen `Graphics2D` er en underklasse af klassen `Graphics`, som tilføjer yderligere grafiske primitiver og attributter.

```
public void paint(Graphics g) {
    Grapichs2D g2d = (Graphics2D) g;
    // use g2d
    ...
}
```

- Grænsefladerne `Shape`, `Stroke`, `Paint` og `Composite`.

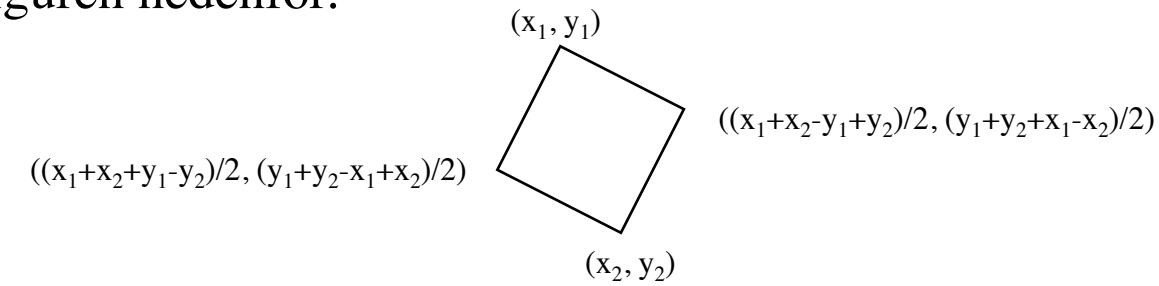
- Klassen `java.awt.geom.AffineTransform` gør det muligt at parallelforskyde, skalere, rotere og klippe figurer.

Ugeseddel 7

12. oktober - 19. oktober

- Læs kapitel 11 i lærebogen (side 547 - 585)

- Løs opgave 9.1 (a), dog kun for diamanter.
  De nødvendige koordinatberegninger er angivet på figuren nedenfor.



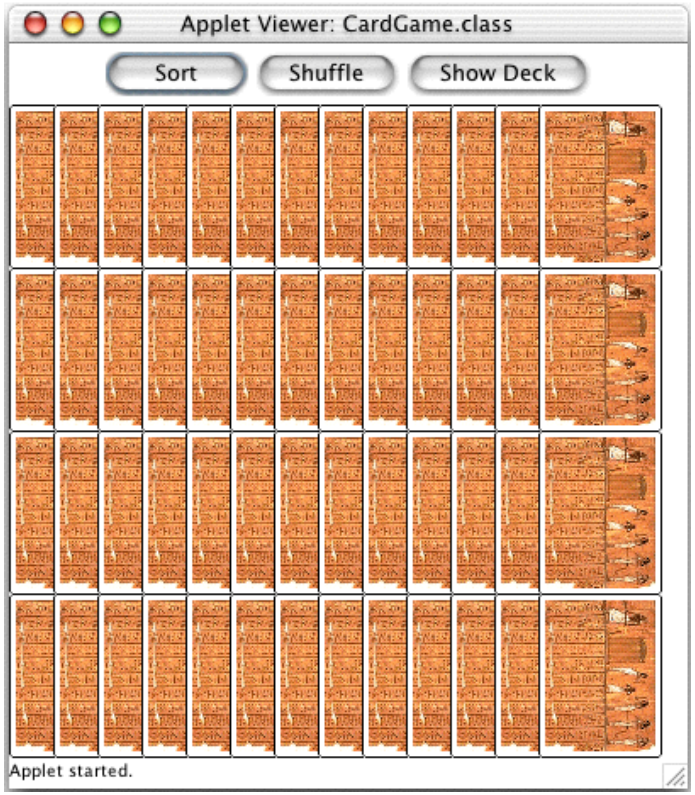
- Løs opgaven på de næste sider.

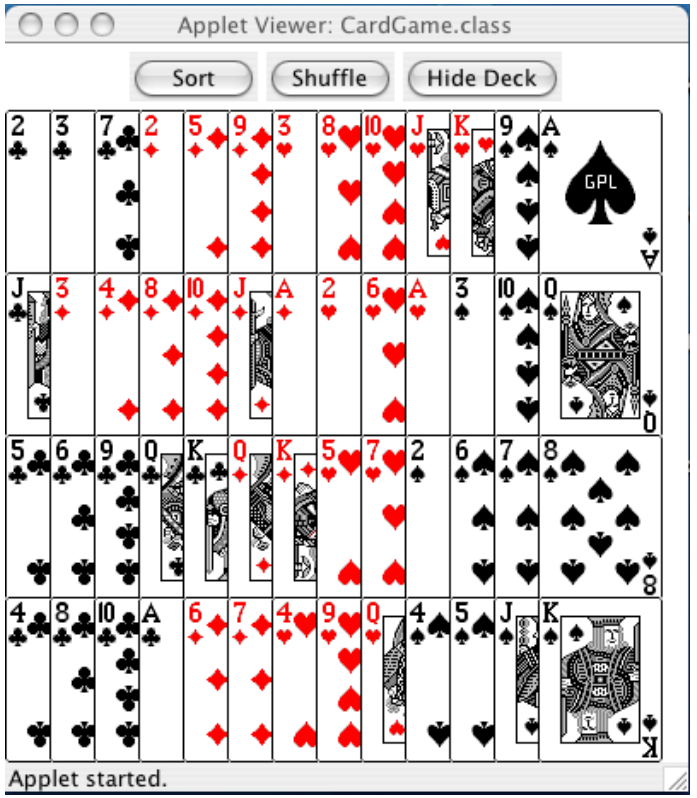
Ekstraopgave 5

Skriv en grafisk applet, der kan blande et spil kort.
En mulig brugergrænseflade er vist på næste side.

Til brug for programmeringen kan klasserne `Card` og `Deck` benyttes. Disse kan sammen med de nødvendige billedfiler (gifs) hentes fra adressen

http://www.akira.ruc.dk/~keld/teaching/OOP_e09/Ekstraopgave5/


Applet Viewer: CardGame.class
Sort
Shuffle
Show Deck
Applet started.


Applet Viewer: CardGame.class
Sort
Shuffle
Hide Deck
GPL
Applet started.