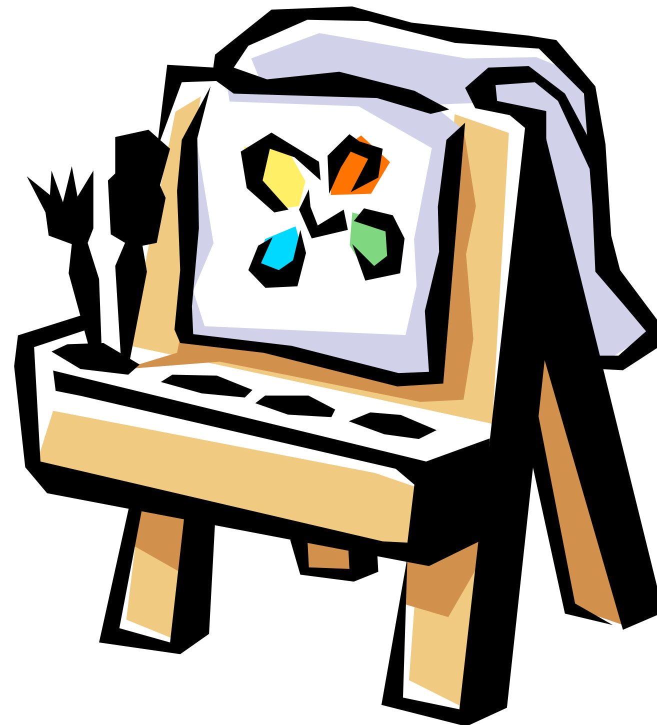
Grafiske brugergrænseflader I



Plan

- Grafiske komponenter
- Layout
- Hændelser og lyttere
- Rammer og dialoger

Nye designmønstre:
Composite
Command

Grafiske brugergrænseflader i Java

Javas GUI framework består af klasser, der kan inddeles i følgende 4 kategorier:

(1) Grafiske komponenter

(2) Layout-managere

(3) Hændelser og lyttere

(4) Grafik-, geometri- og billedklasser

(1) Grafiske komponenter (widgets, dimser)

look: udseende
feel: opførsel ved brugerinput

Eksempler:

`Button`, `Label`,
`Checkbox`, `Scrollbar`,
`Frame`, `Dialog`

(2) Layout-managere

Angiver strategier for udlægning af komponenter i et vindue

Eksempler:

`FlowLayout`, `GridLayout`, `BorderLayout`

(3) Hændelser og lyttere

Hændelser repræsenterer brugerhandlinger.
Lyttere modtager og behandler hændelserne.

Eksempler:

Type	Lytter
`MouseEvent`	`MouseListener`
`KeyEvent`	`KeyListener`
`ActionEvent`	`ActionListener`

(4) Grafik-, geometri- og billedklasser

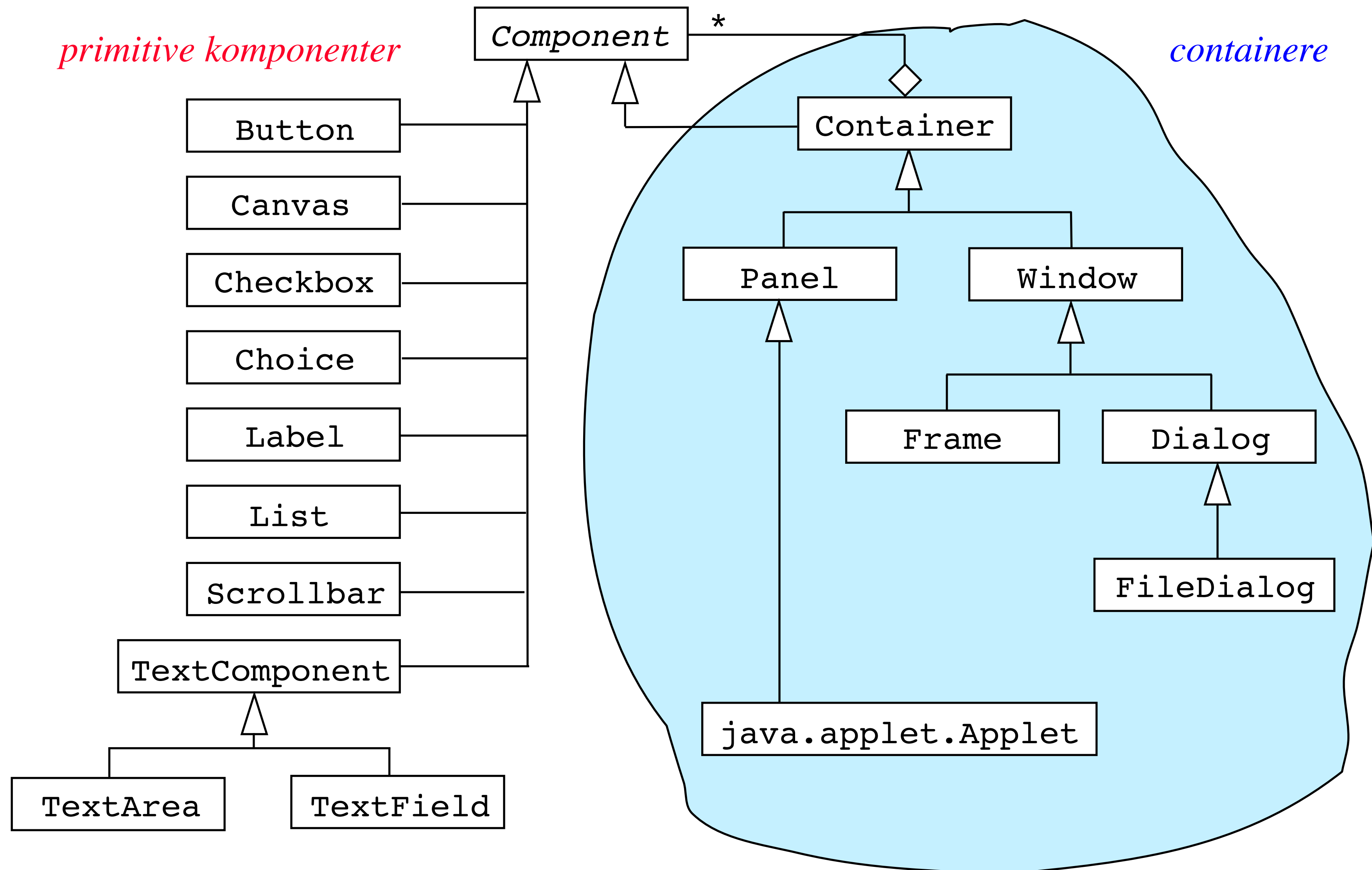
Eksempler:

Grafik	Geometri	Billede
`Graphics`	`Dimension`	`Image`
`Font`	`Rectangle`	`Icon`
`Color`	`Point`	

AWT tilbyder de grundlæggende faciliteter til GUI-konstruktion.

Swing er en udvidelse af AWT (flere komponenter og vægt på platformsuafhængighed).

Komponenthierarkiet i AWT



Designmønsteret Composite

Kategori:

Strukturelt designmønster

Hensigt:

At arrangere objekter i træstrukturer for at repræsentere et hierarki af enkeltdele eller helheder. Composite gør det muligt for klienter at behandle individuelle objekter og samlinger af objekter på en ensartet måde.

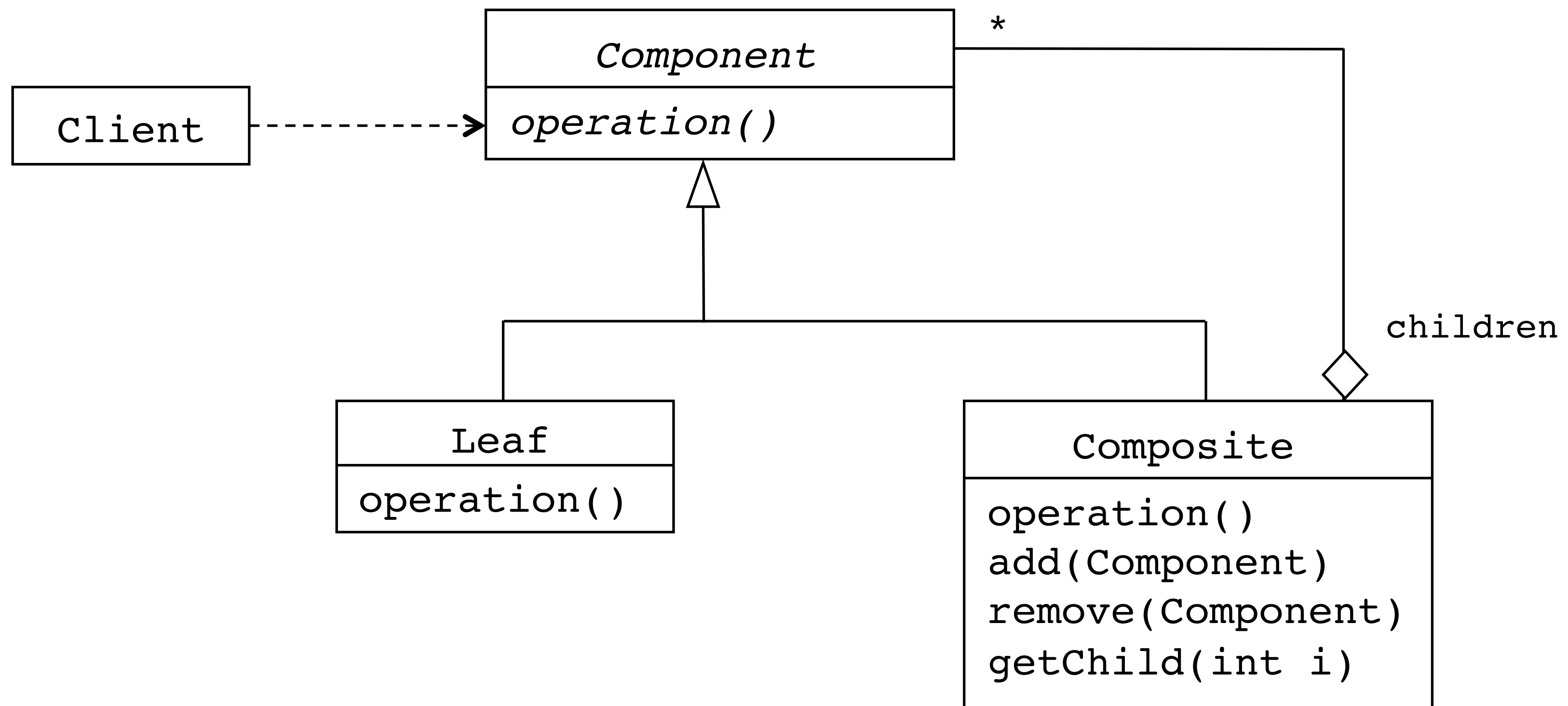
Anvendelse:

- For at repræsentere et del-helheds-hierarki af objekter.
- Når det ønskes, at klienter skal ignorere forskellen imellem samlinger af objekter og enkeltobjekter.

Designmønsteret Composite

(fortsat)

Struktur:



Designmønsteret Composite
(fortsat)

Deltagere:

Component (f.eks. `Component`), der definerer grænsefladen for objekter i sammensætningen

Leaf (f.eks. `Button`, `Label` og `TextBox`), der definerer de primitive objekter i sammensætningen.

Composite (f.eks. `Container` og `Panel`), der repræsenterer sammensætninger af objekter.

Client, som arbejder med objekterne igennem *Component*-grænsefladen.

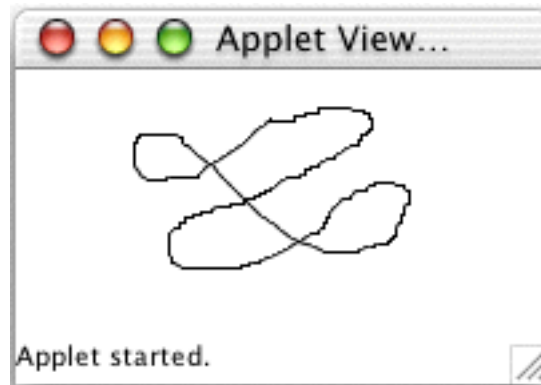
AWT-komponenter

`Button`

Knap, der reagerer på museklik

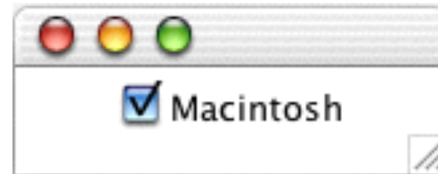
`Canvas`

Rektangulært område, som kan benyttes til at tegne



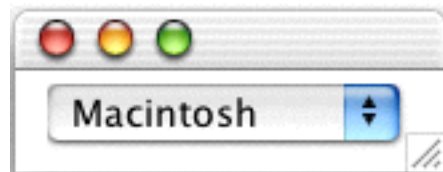
`Checkbox`

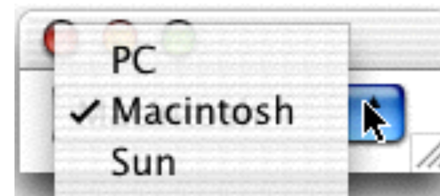
Komponent, der kan være enten “tændt” eller “slukket”



`Choice`

Popup-menu med tekstvalg, hvorfra brugeren kan vælge



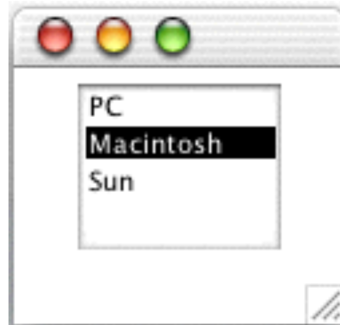


`Label`

Passiv komponent med tekst

`List`

Rullende liste af tekstemner, hvorfra brugeren kan vælge

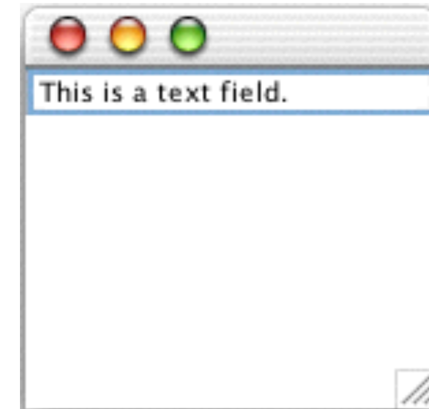


`Scrollbar`

Rullebjælke

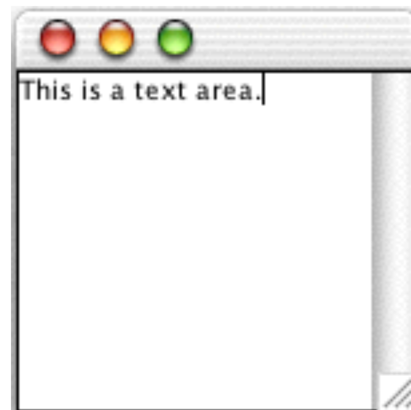
`TextField`

Tekstfelt til redigering af en enkelt linje



`TextArea`

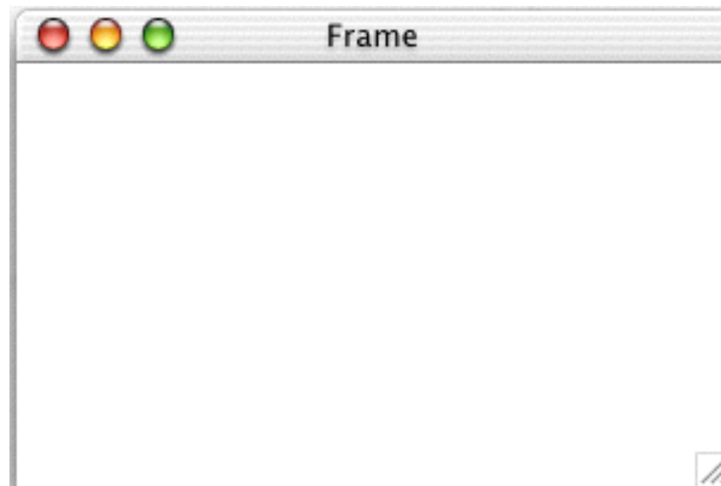
Rektangulært tekstområde, der tillader redigering af flere linjer



AWT-containere

`Frame`

Topniveau-vindue med en titel og en rand



`Window`

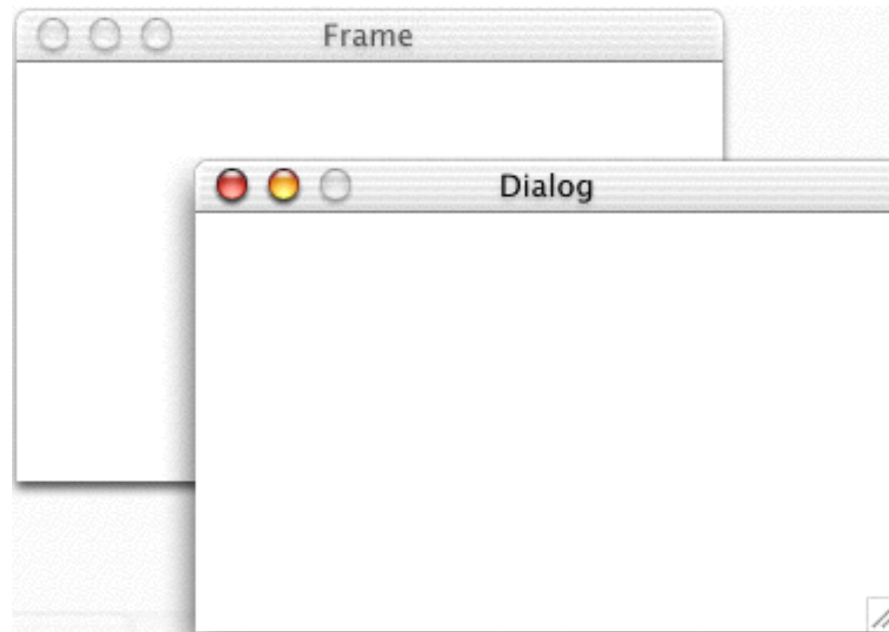
Topniveau-vindue uden titel og rand (bruges sjældent)

`Panel`

Container uden titel og rand

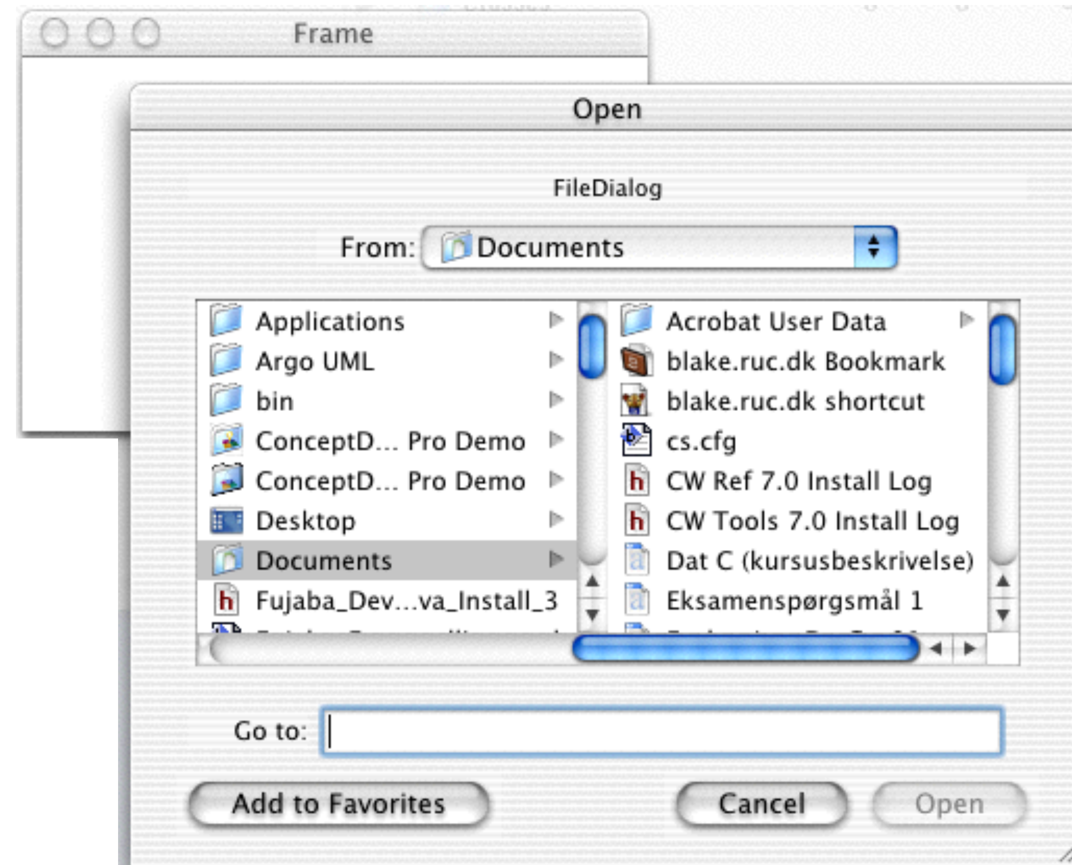
`Dialog`

Vindue, der modtager input fra brugeren



`FileDialog`

Dialogvindue, hvorfra brugeren kan vælge en fil



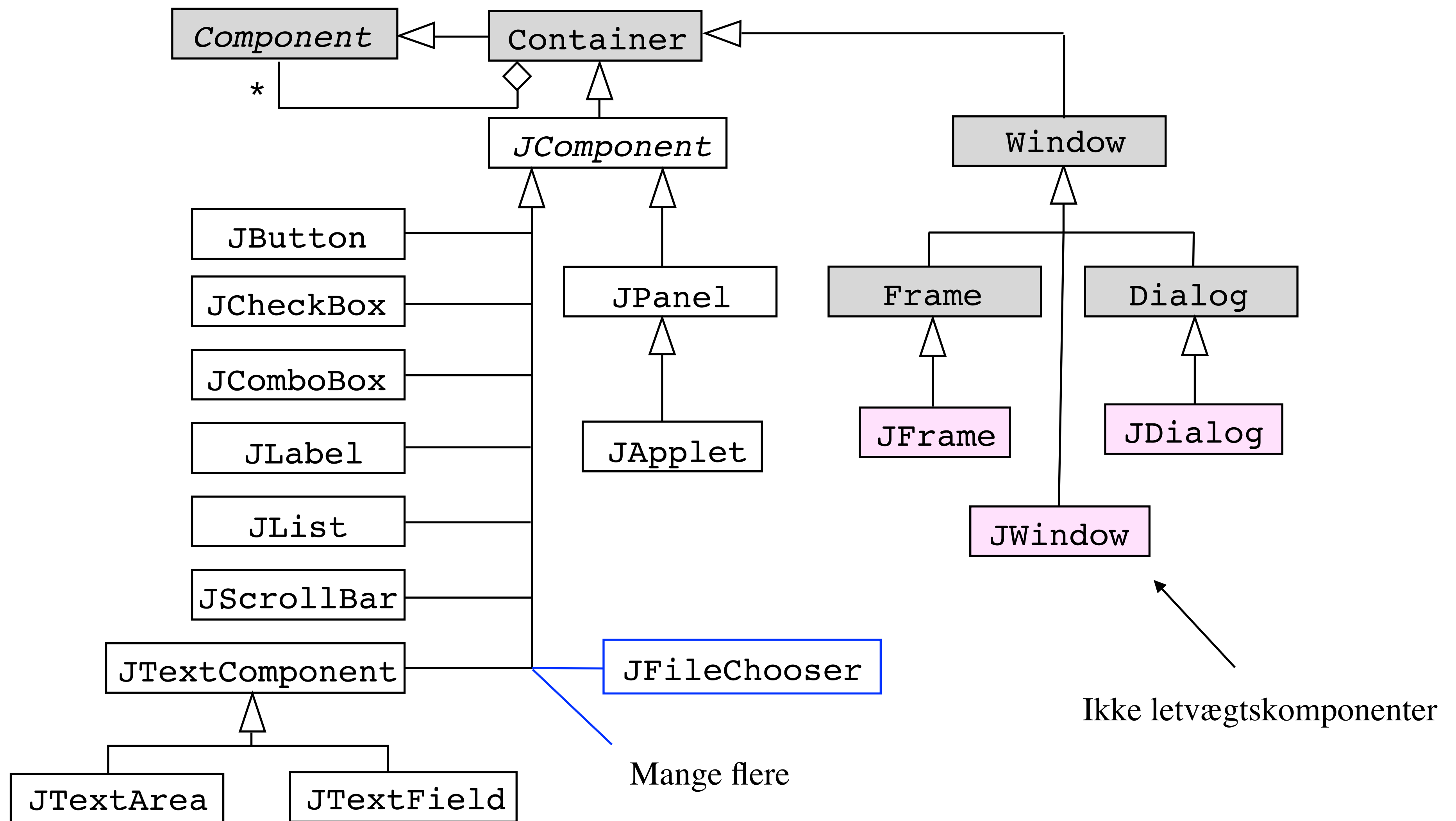
Swing
en udvidelse af AWT

- Flere komponenter

- Vægt på platformsuafhængighed
  Swing: **letvægt**skomponenter (skrevet i Java)
  AWT: **sværvægt**skomponenter (kommunikerer med det underliggende vinduessystem via *peer*-komponenter)

- Større effektivitet
  Der skal ikke skabes peer-komponenter

Swings sidestykke til AWT-komponenterne



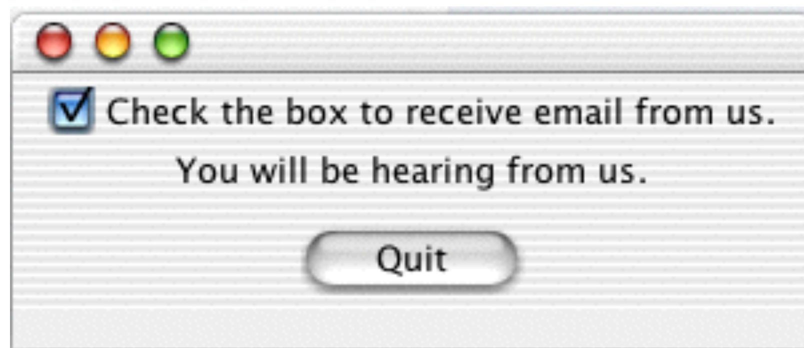
Swing-komponenter

`JButton`

Knap, der reagerer på museklik (kan indeholde komponenter)

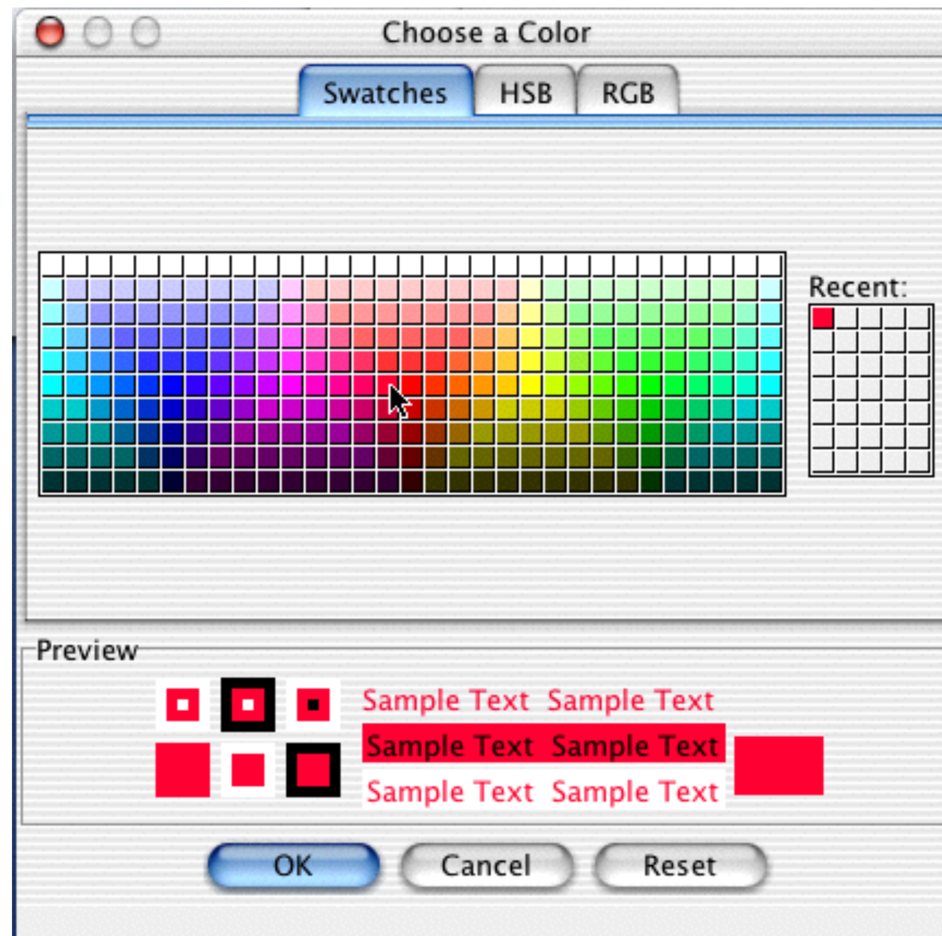
`JCheckBox`

Komponent, der kan være enten "tændt" eller "slukket"



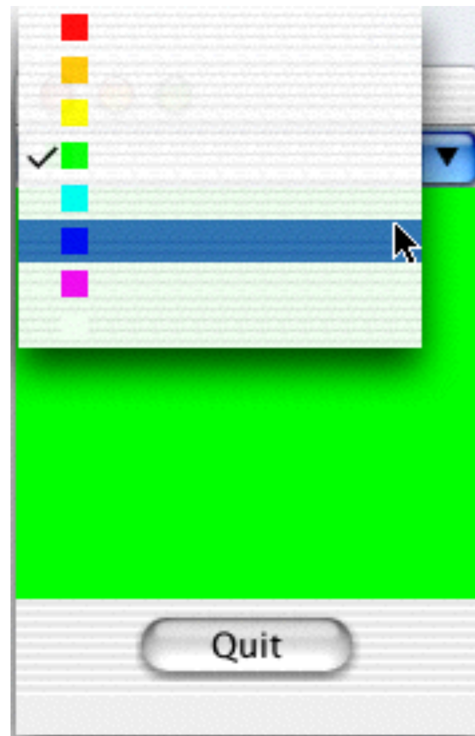
`JColorChooser`

Komponent, der tillader brugeren at vælge en farve



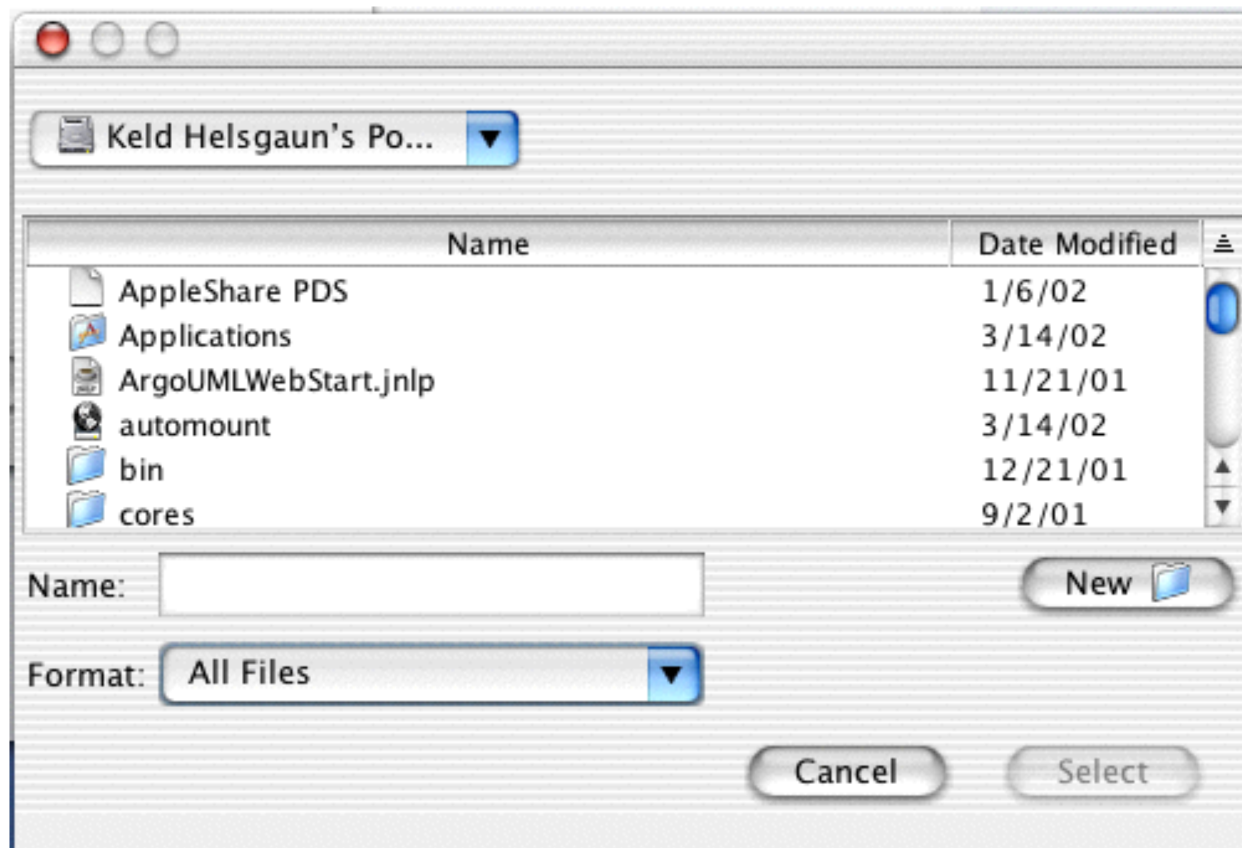
`JComboBox`

Popup-menu med valgbare emner



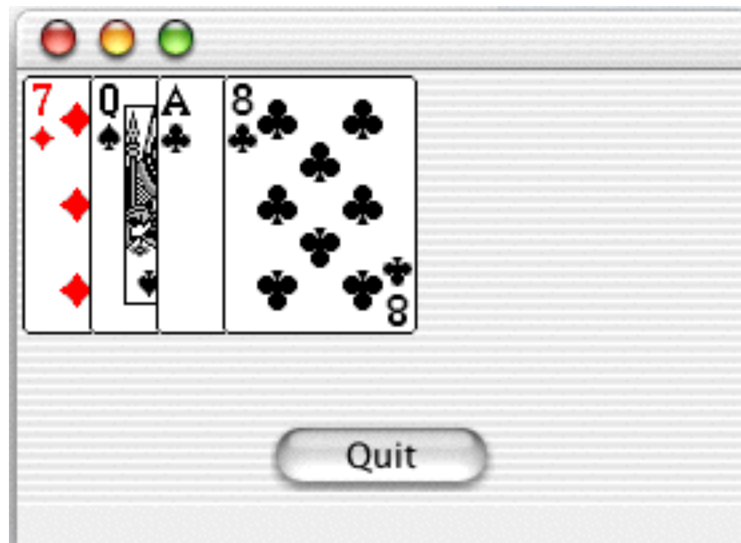
`JFileChooser`

Komponent, der tillader brugeren at vælge en fil



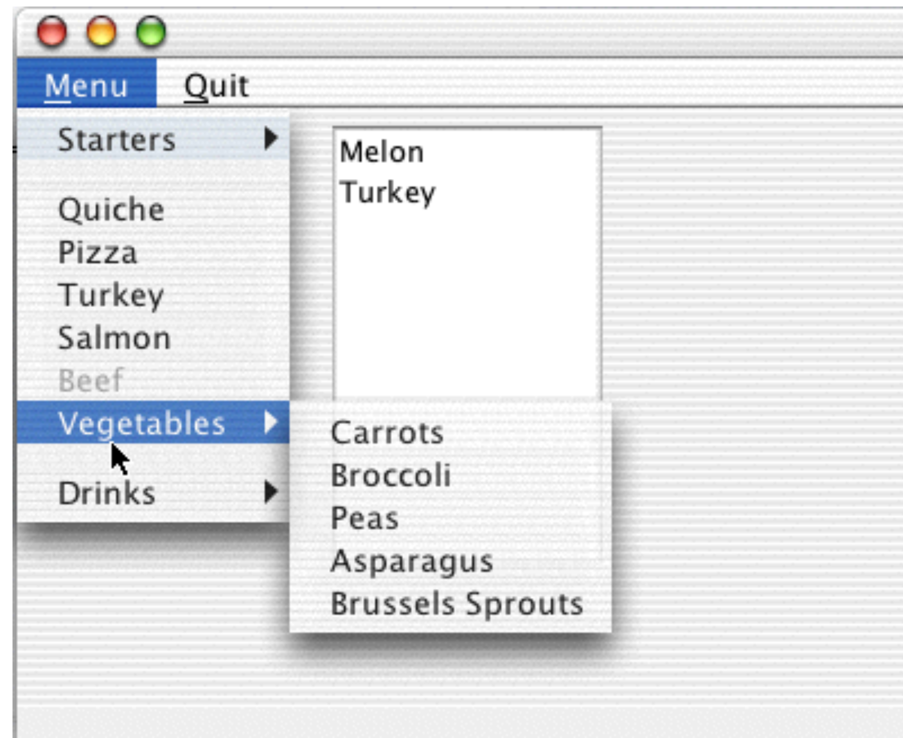
`JLayeredPane`

Container, der viser sine børnekomponenter i lag



`JMenu`

Rulned-menu i en menubjælke (kan indholde undermenuer)

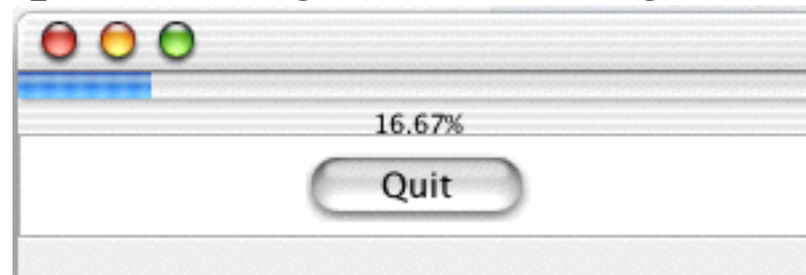


`JOptionPane`

Simpel dialogboks

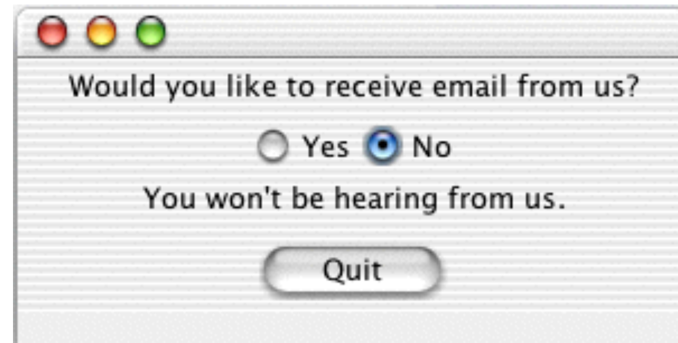
`JProgessBar`

Komponent til grafisk visning af en værdi



`JRadioButton`

Knap til skift imellem to tilstande

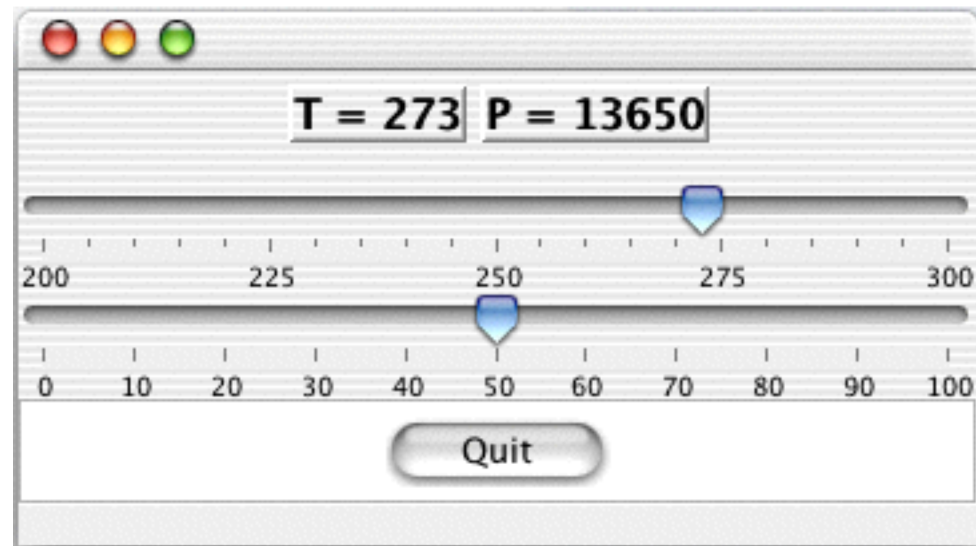


`JScrollPane`

Container med mulighed for “rulning” af sin barnekomponent

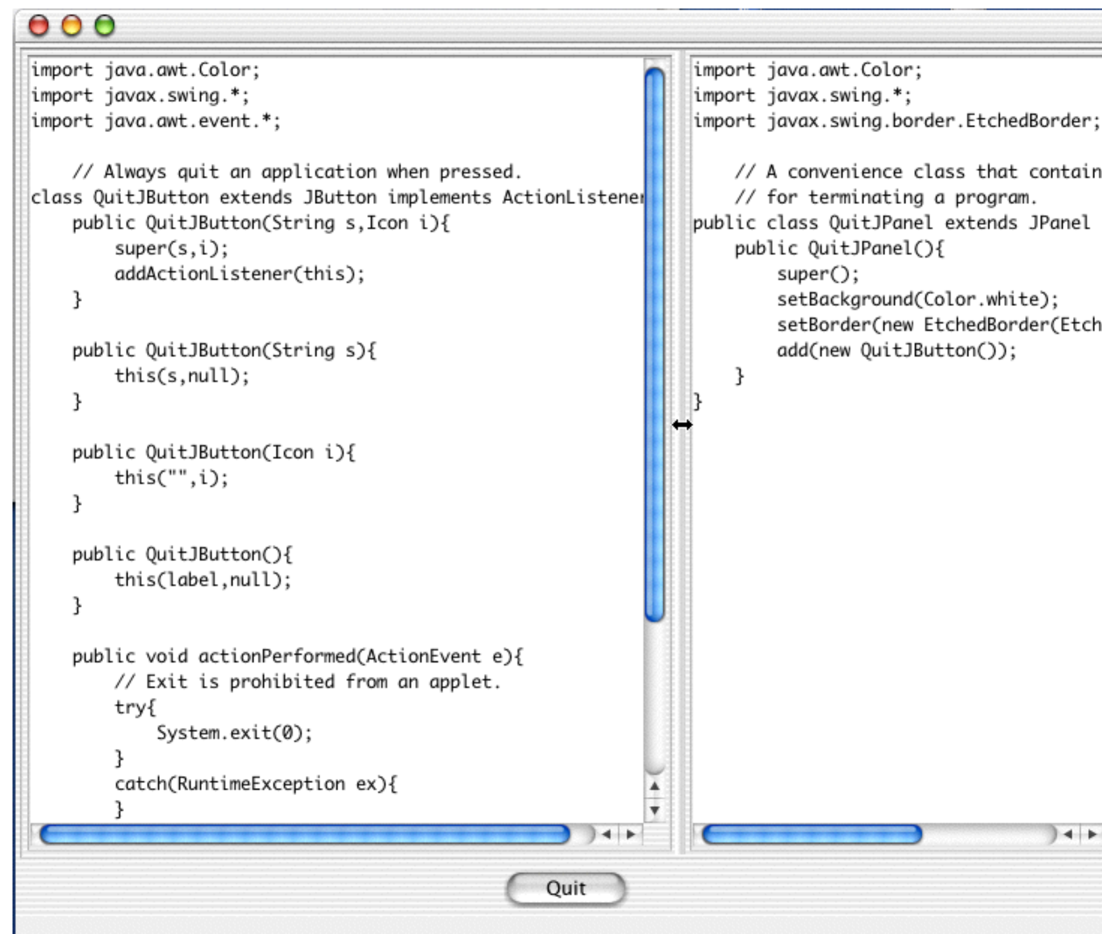
`JSlider`

Komponent, der gør det muligt at ændre en værdi ved at trække i en dup



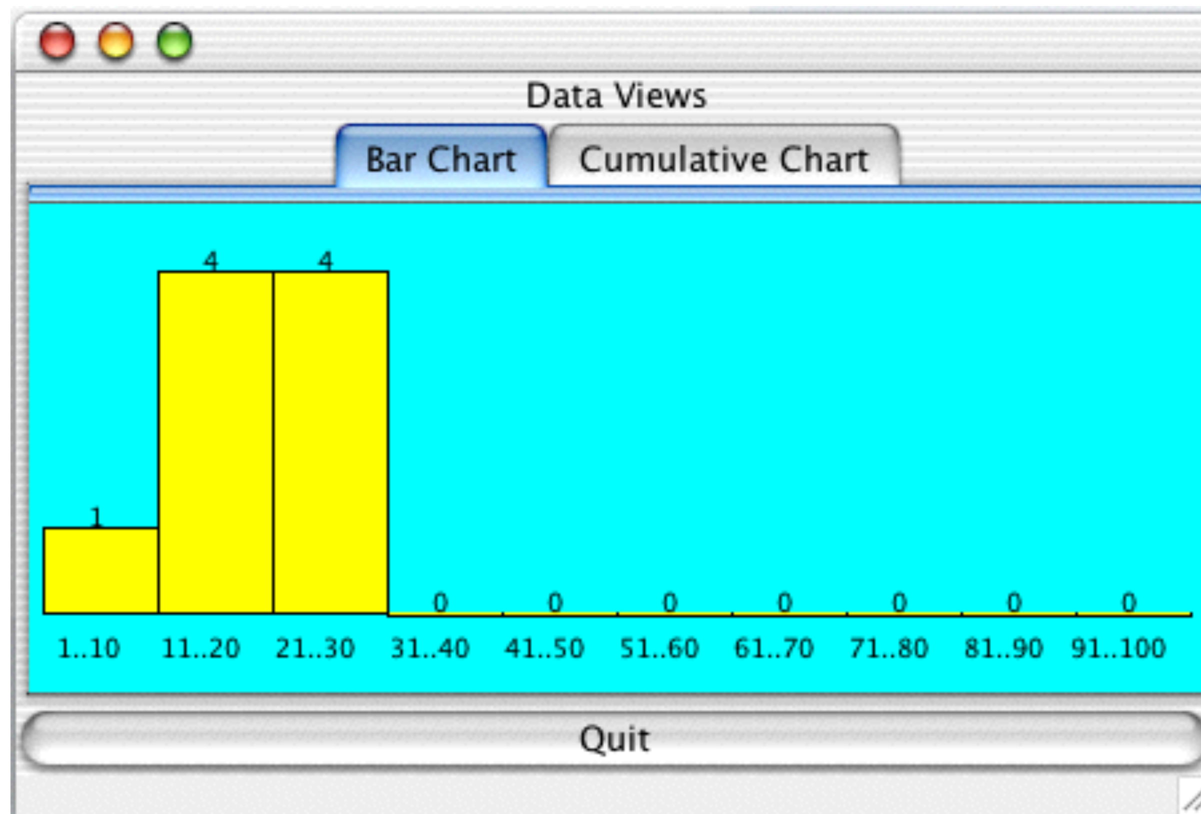
`JSplitPane`

Container, der splittes op vandret eller lodret ved visning af sine to børnekomponenter



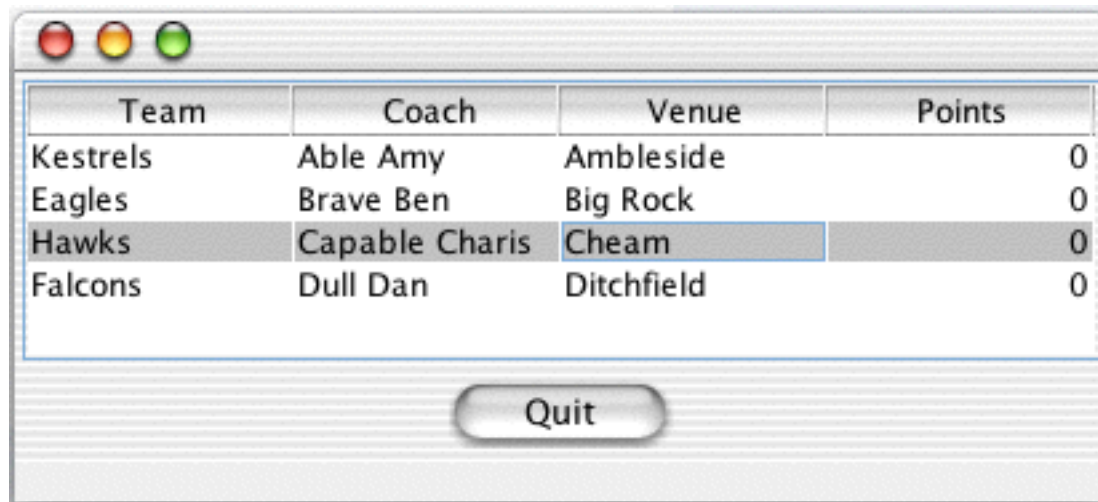
`JTabbedPane`

Container, der ved tryk på en fane kan vise en af sine børnekomponenter



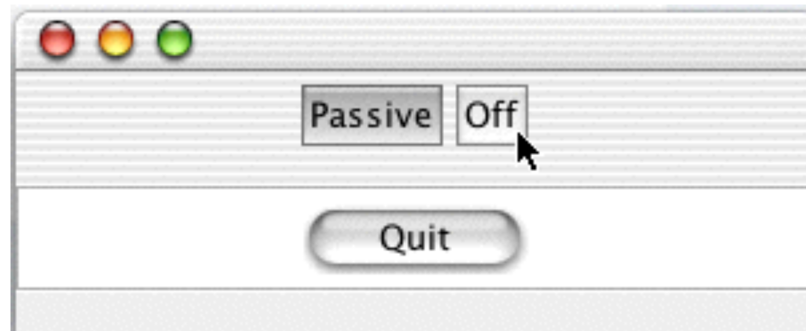
`JTable`

Komponent til visning af tabellagte data



`JToggleButton`

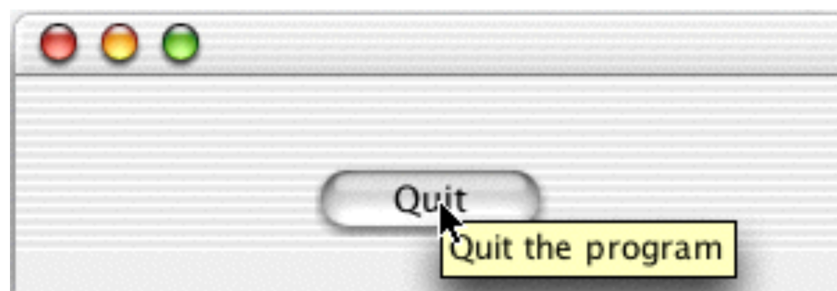
Knap, der kan skifte imellem at være valgt og ikke-valgt



Underklasser:
`JCheckBox`,
`JRadioButton`

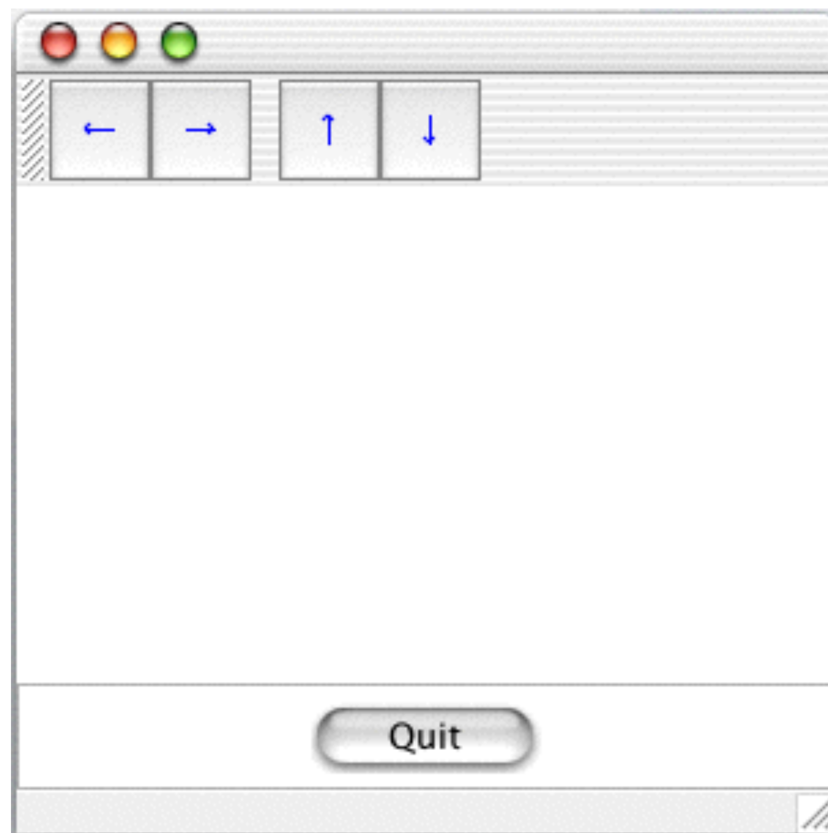
`JToolTip`

Komponent til visning af et tip til brug af en komponent



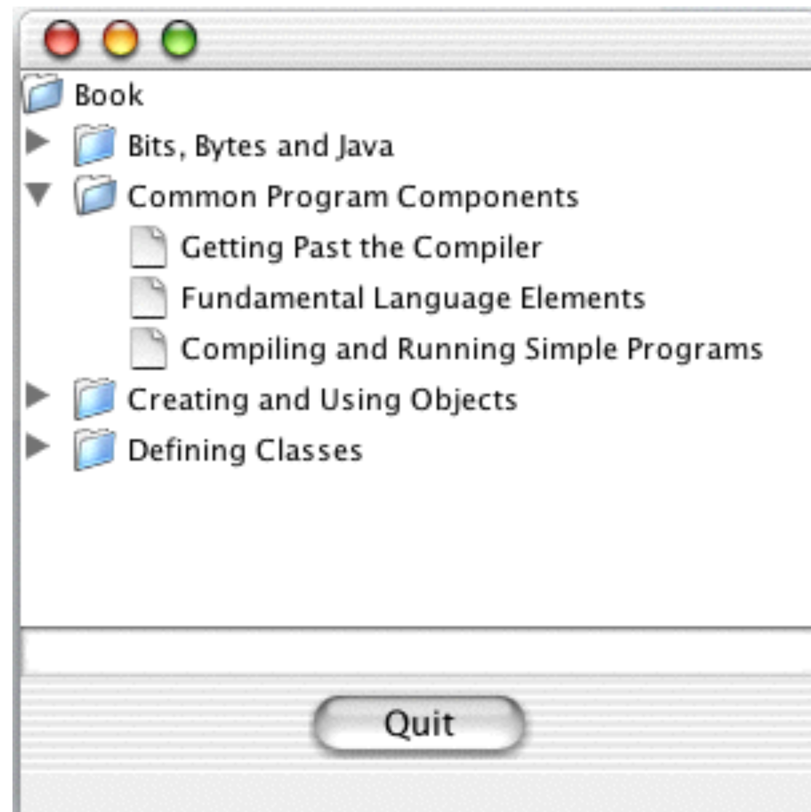
`JToolBar`

Lodret eller vandret værktøjslinje



`JTree`

Komponent til visning af hierarkisk strukturerede data



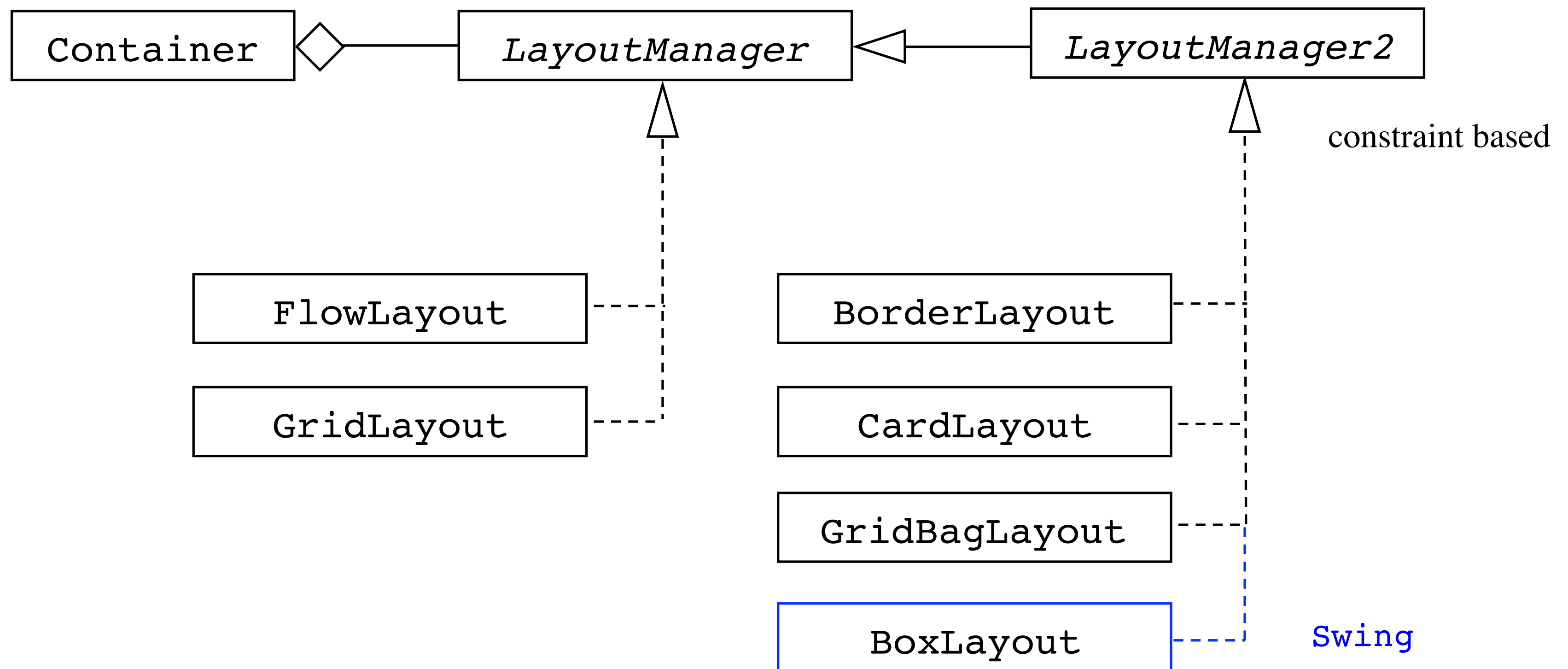
Layout

Enhver container er forsynet med en **layout-manager**, der definerer en strategi for udlægning af containerens komponenter.

Udlægningen sker ud fra komponenternes *relative* positioner.

De relative positioner kan enten være specificeret eksplicit, ved hjælp af positionelle betingelser (constraints), eller implicit, ved rækkefølgen, hvori de er lagt i containeren.

Layout-managere



Bemærk brugen af designmønsteret Strategy

Metoder i `Container`

void setLayout(LayoutManager lm)

void add(Component comp)

void add(Component comp, Object constraints)

FlowLayout

Komponenterne placeres fra venstre mod højre i den orden, de er blevet indsat i containeren.

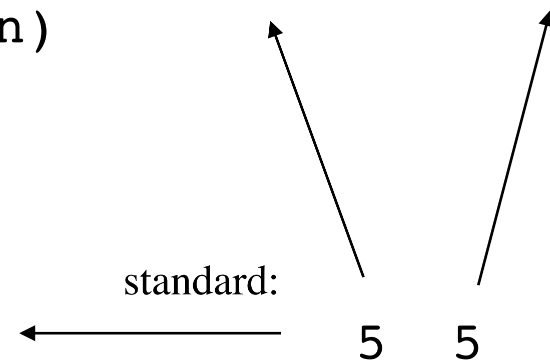
Hvis containeren ikke er bred nok til at indeholde alle sine komponenter, placeres komponenterne i rækker.

Konstruktører:

```
FlowLayout(int align, int hGap, int vGap)
FlowLayout(int align)
FlowLayout()
```

Justering (alignment):

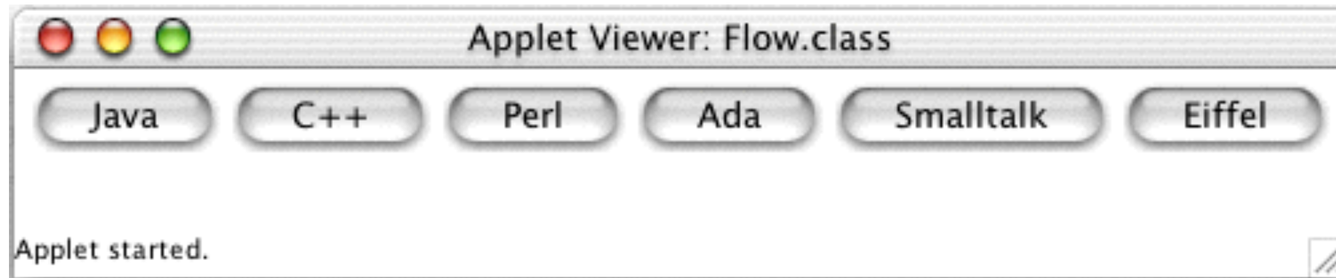
```
FlowLayout.LEFT
FlowLayout.CENTER
FlowLayout.RIGHT
```



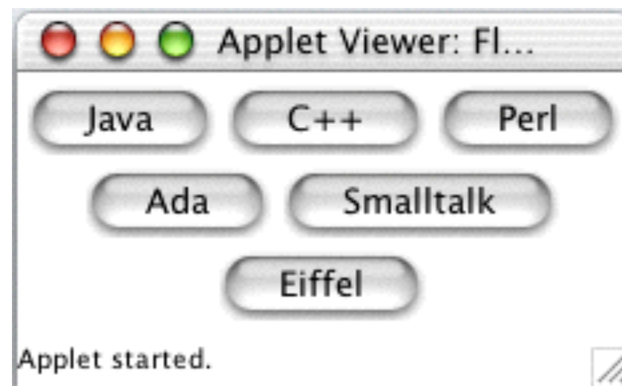
Eksempel på brug af `FlowLayout`

```
import java.awt.*;
import java.applet.Applet;

public class Flow extends Applet {
    public Flow() {
        setLayout(new FlowLayout());
        add(new Button("Java"));
        add(new Button("C++"));
        add(new Button("Perl"));
        add(new Button("Ada"));
        add(new Button("Smalltalk"));
        add(new Button("Eiffel"));
    }
}
```

Kørsler med `Flow`



`width=500 height=60`



`width=230 height=100`

GridLayout

Komponenterne placeres i et rektangulært net.
Alle komponenter tildeles samme plads.

Som standard er der 1 række.

Konstruktører:

```
GridLayout(int r, int c, int hGap, int vGap)
GridLayout(int r, int c)
GridLayout()
```

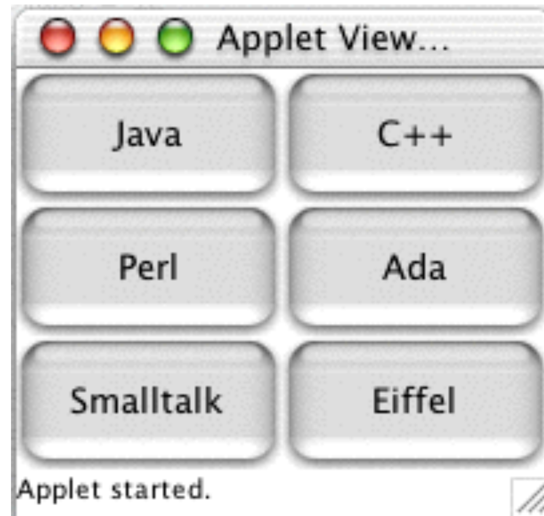
standard: 0 0

`r`: antal rækker hvis 0, et variabelt antal
`c`: antal søjler hvis 0, et variabelt antal

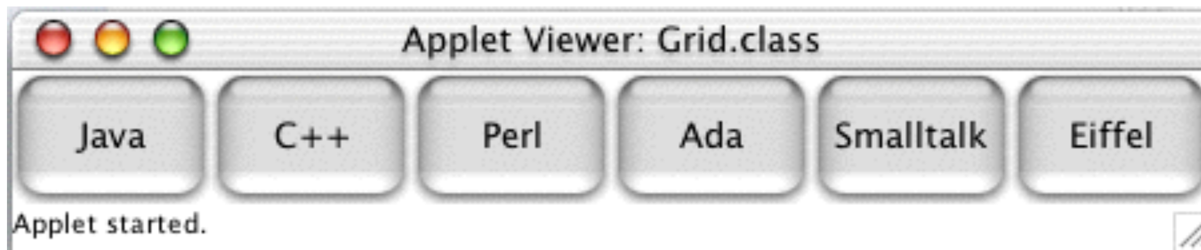
Eksempel på brug af **`GridLayout`**

```
public class Grid extends Applet {
    public void init() {
        int row = 0, col = 0;
        String att = getParameter("row");
        if (att != null) row = Integer.parseInt(att);
        att = getParameter("col");
        if (att != null) col = Integer.parseInt(att);
        if (row == 0 && col == 0) {
            row = 3; col = 2;
        }
        setLayout(new GridLayout(row, col));
        add(new Button("Java"));
        add(new Button("C++"));
        add(new Button("Perl"));
        add(new Button("Ada"));
        add(new Button("Smalltalk"));
        add(new Button("Eiffel"));
    }
}
```

Kørsler med **Grid**



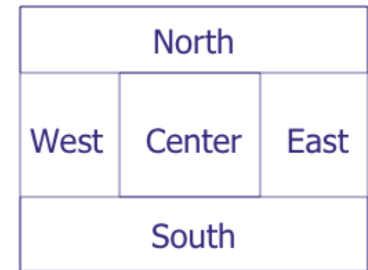
`row=3 col=2`



`row=1 col=0`

`row=0 col=1`

BorderLayout



Komponenterne placeres i fem områder:

`NORTH`, `SOUTH`, `EAST`, `WEST`, `CENTER`

Konstruktører:

```
BorderLayout(int hGap, int vGap)
BorderLayout()
```

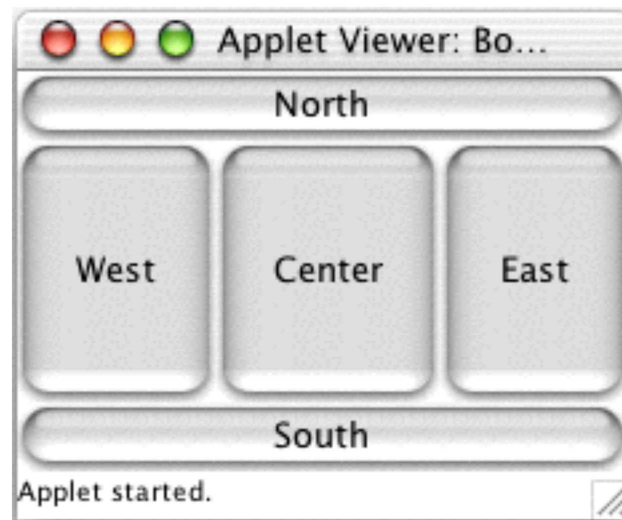
Indsættelse i containeren sker ved brug af en af følgende "constraints":

```
BorderLayout.NORTH
BorderLayout.SOUTH
BorderLayout.EAST
BorderLayout.WEST
BorderLayout.CENTER
```

Eksempel på brug af **`BorderLayout`**

```
public class Border extends Applet {
    public Border() {
        setLayout(new BorderLayout());
        add(new Button("South"),  BorderLayout.SOUTH);
        add(new Button("North"),  BorderLayout.NORTH);
        add(new Button("East"),   BorderLayout.EAST);
        add(new Button("West"),   BorderLayout.WEST);
        add(new Button("Center"), BorderLayout.CENTER);
    }
}
```

Kørsel med `Border`



`width=230 height=150`

CardLayout

Komponenterne placeres oven på hinanden (som spillekort).

Konstruktører:

```
CardLayout(int hGap, int vGap)
CardLayout()
```

Indsættelse i containeren sker ved brug af `add` med 2 parametre:

```
add(Component comp, Object constraints)
```

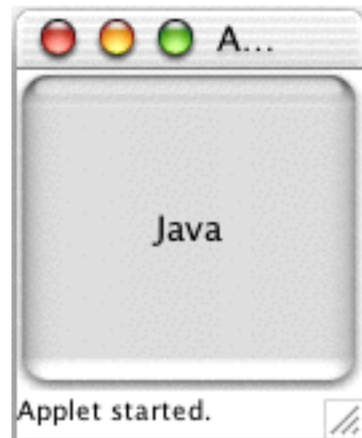
eller

```
add(String name, Component comp)
```

Eksempel på brug af `CardLayout`

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;

public class Card extends Applet implements ActionListener {
    CardLayout cards;

    public Card() {
        cards = new CardLayout();
        setLayout(cards);
        Button b1 = new Button("Java");
        b1.addActionListener(this);
        add("Java", b1);
        Button b2 = new Button("Perl");
        b2.addActionListener(this);
        add("Perl", b2);
    }

    public void actionPerformed(ActionEvent e) {
        cards.next(this);
    }
}
```

Kørsel med `Card`



`width=200 height=150`

BoxLayout

`javax.swing`

Komponenterne placeres i enten 1 række eller 1 søjle.

Konstruktør:

```
BoxLayout(Container target, int axis)
```

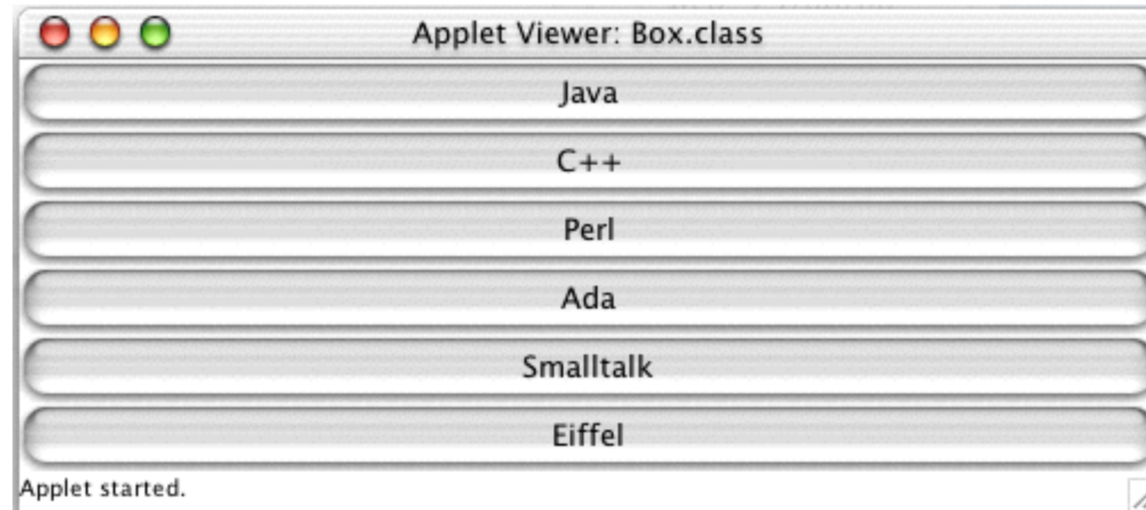
hvor `axis` er en af følgende konstanter:

```
BoxLayout.X_AXIS
BoxLayout.Y_AXIS
```

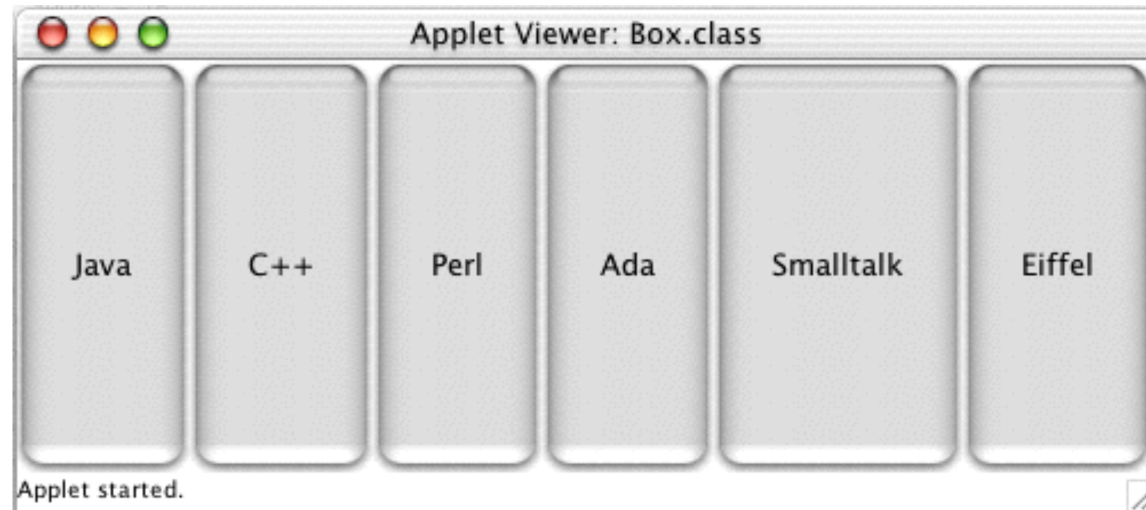
Eksempel på brug af `BoxLayout`

```
import java.awt.*;
import java.applet.Applet;
import javax.swing.*;

public class Box extends Applet {
    public Box() {
        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
        add(new Button("Java"));
        add(new Button("C++"));
        add(new Button("Perl"));
        add(new Button("Ada"));
        add(new Button("Smalltalk"));
        add(new Button("Eiffel"));
    }
}
```

Kørsler med **Box**



Y_AXIS



X_AXIS

width=500 height=180

GridBagLayout

Komponenterne placeres i et rektangulært net.
Komponenter tildeles plads ud fra givne betingelser (constraints).

Konstruktør:

```
GridBagLayout()
```

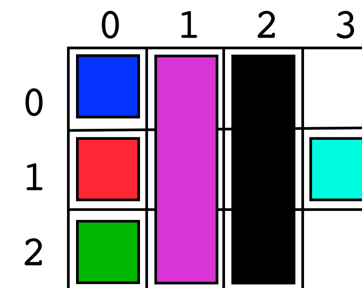
Betingelser angives ved brug af et `GridBagConstraints`-objekt.

Eksempel på brug af `GridBagLayout`

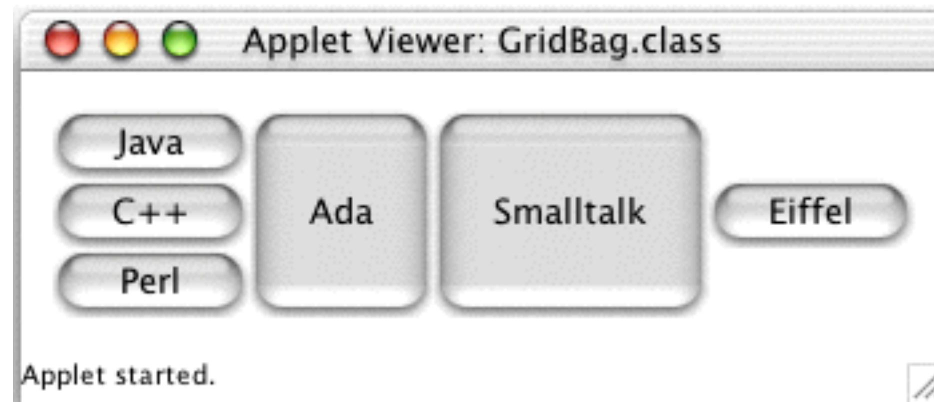
```
public class GridBag extends Applet {
    private GridBagLayout gbLayout;
    private GridBagConstraints gbConstraints;

    private void addComponent(Component c,
                              int row, int column,
                              int width, int height) {
        gbConstraints.gridx = column;
        gbConstraints.gridy = row;
        gbConstraints.gridwidth = width;
        gbConstraints.gridheight = height;
        gbLayout.setConstraints(c, gbConstraints);
        add(c);
    }

    public void init() { ... }
}
```

fortsættes

```
public void init () {
    gbLayout = new GridBagLayout();
    gbConstraints = new GridBagConstraints();
    setLayout(gbLayout);
    gbConstraints.fill = GridBagConstraints.BOTH;
    addComponent(new Button("Java"),      0, 0, 1, 1);
    addComponent(new Button("C++"),       1, 0, 1, 1);
    addComponent(new Button("Perl"),      2, 0, 1, 1);
    addComponent(new Button("Ada"),       0, 1, 1, 3);
    addComponent(new Button("Smalltalk"), 0, 2, 1, 3);
    addComponent(new Button("Eiffel"),    1, 3, 1, 1);
}
```



Kørsel med `GridBag`



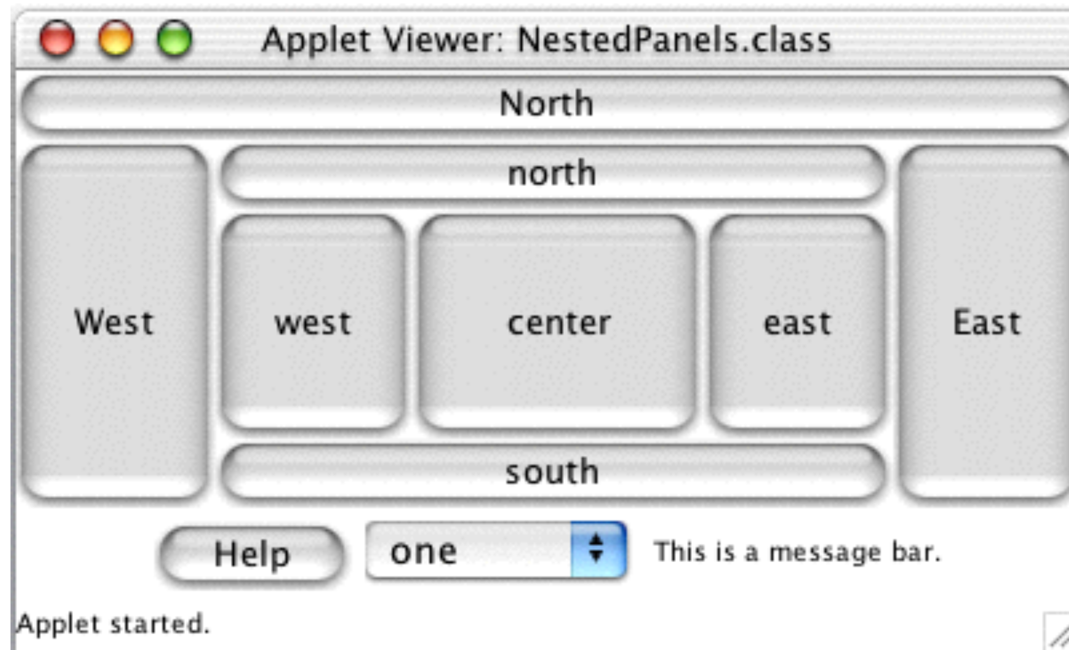
Indlejrede paneler

```
public class NestedPanels extends Applet {
    public NestedPanels () {
        Panel center = new Panel();
        center.setLayout(new BorderLayout());
        center.add(new Button("south"),  BorderLayout.SOUTH);
        center.add(new Button("north"),  BorderLayout.NORTH);
        center.add(new Button("east"),   BorderLayout.EAST);
        center.add(new Button("west"),   BorderLayout.WEST);
        center.add(new Button("center"), BorderLayout.CENTER);
```

fortsættes

```
        Panel south = new Panel();
        south.setLayout(new FlowLayout());
        south.add(new Button("Help"));
        choice = new Choice();
        choice.addItem("one");
        choice.addItem("two");
        choice.addItem("three");
        choice.addItem("four");
        choice.addItem("five");
        south.add(choice);
        messageBar = new Label("This is a message bar.");
        south.add(messageBar);

        setLayout(new BorderLayout());
        add(new Button("North"), BorderLayout.NORTH);
        add(new Button("East"),  BorderLayout.EAST);
        add(new Button("West"),  BorderLayout.WEST);
        add(south,               BorderLayout.SOUTH);
        add(center,              BorderLayout.CENTER);
    }

    protected Label messageBar;
    protected Choice choice;
}
```

Kørsel med `NestedPanels`



`width=400 height=200`

Hændelser og lyttere

GUI-komponenter kommunikerer med de øvrige dele af et program igennem *hændelser*.

Kilden til en hændelse er den komponent, hvorfra hændelsen stammer.

En *lytter* er et objekt, som modtager og behandler en hændelse.

Lyttere må registreres hos deres kilder, før de kan modtage hændelser fra dem.

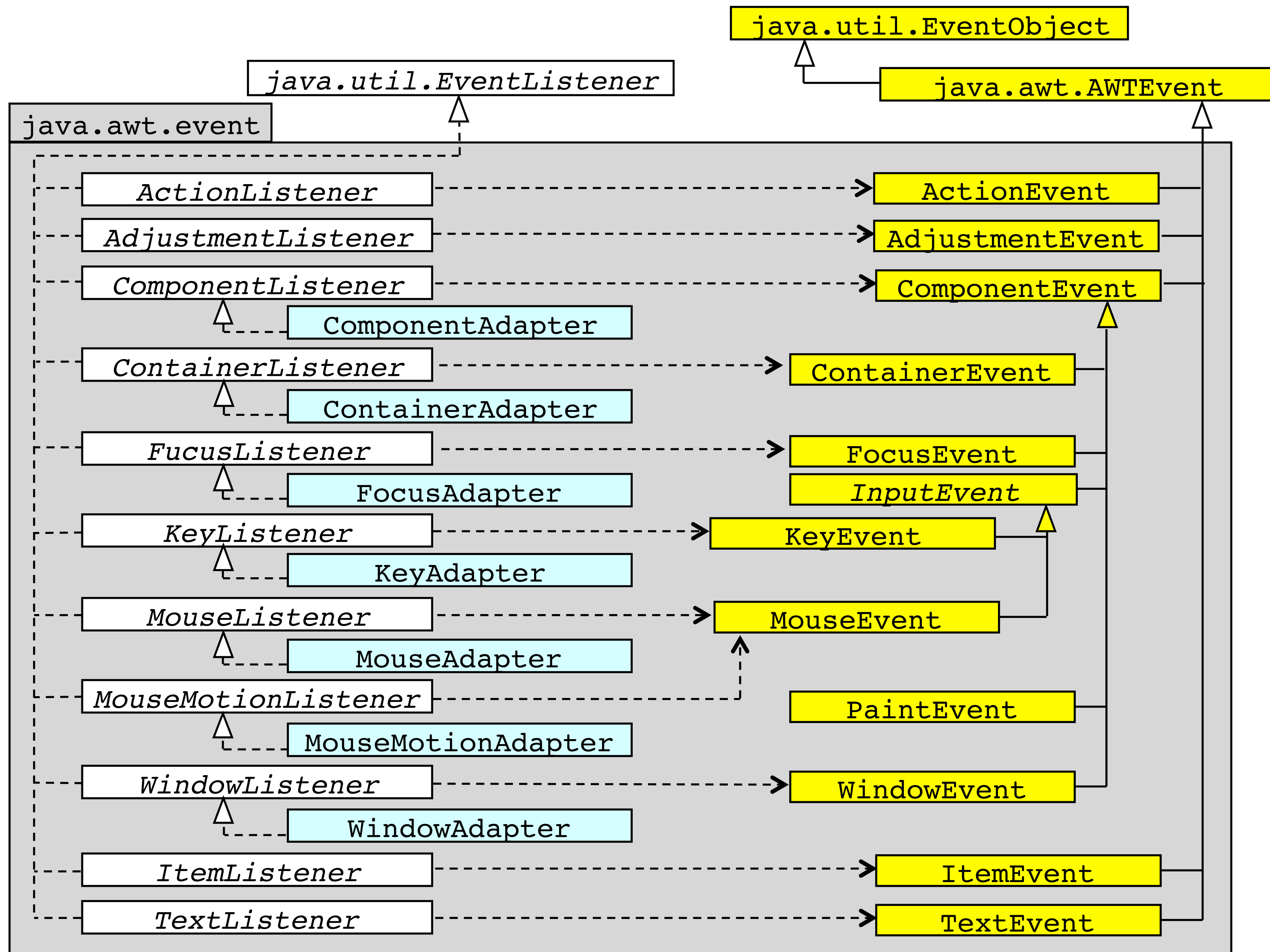
Lyttere og deres kilder

Lytter	Kilde

`ActionListener`	`Button`
	`TextField`
	`List`

`ItemListener`	`Checkbox` `Choice`
`AdjustmentListener`	`Scrollbar`

`MouseListener`
`MouseMotionListener`
`FocusListener`
`KeyListener`
	alle komponenter



java.util.EventObject
java.awt.AWTEvent
java.util.EventListener
java.awt.event
ActionListener
ActionEvent
AdjustmentListener
AdjustmentEvent
ComponentListener
ComponentEvent
ComponentAdapter
ContainerListener
ContainerEvent
ContainerAdapter
FucusListener
FocusEvent
FocusAdapter
InputEvent
KeyListener
KeyEvent
KeyAdapter
MouseListener
MouseEvent
MouseAdapter
MouseMotionListener
PaintEvent
MouseMotionAdapter
WindowListener
WindowEvent
WindowAdapter
ItemListener
ItemEvent
TextListener
TextEvent

Programmering med hændelser

(1) Bestem de hændelser, der er involveret, og find de tilsvarende lytter-grænseflader.

(2) Erklær lytter-klasser, der implementerer disse grænseflader.

(3) Skab instanser af de komponenter, der er kilder til hændelserne.

(4) Skab instanser af lytter-klasserne og registrer dem hos deres kilder.

Eksempel

Tryk på en knap

(1) Hændelsesklassen er `ActionEvent`.

(2) Den tilsvarende lytter-grænseflade er `ActionListener`.

```
class MyButtonHandler implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        // ... handle button click
    }
}
```

(3)

```
Button button1 = new Button("One");
```

(4)

```
MyButtonHandler handler = new MyButtonHandler();
button1.addActionListener(handler);
```

Eksempelprogram

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

class MyButtonHandler implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        Button source = (Button) event.getSource();
        System.out.println(source.getLabel() + " was clicked");
        if (source.getLabel().equals("Quit"))
            System.exit(0);
    }
}
```

fortsættes

```
class ButtonTest extends Frame {
    ButtonTest() {
        setLayout(new FlowLayout());
        MyButtonHandler handler = new MyButtonHandler();
        Button button1 = new Button("One");
        add(button1);
        button1.addActionListener(handler);
        Button button2 = new Button("Two");
        add(button2);
        button2.addActionListener(handler);
        Button quitButton = new Button("Quit");
        add(quitButton);
        quitButton.addActionListener(handler);
        pack();        // Minimize window. Necessary!
        setVisible(true);
    }

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

Enhver klasse kan være lytter

```
class ButtonTest extends Frame
                 implements ActionListener {
    ButtonTest() {
        // ...
        Button button1 = new Button("One");
        add(button1);
        button1.addActionListener(this);
        // ...
    }

    public void actionPerformed(ActionEvent event) {
        // ...
    }

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

Fleksibilitet

- En kilde have flere flere lyttere
- En lytter kan lytte på flere kilder
- En lytter kan lytte på flere typer af hændelser (ved at implementere de relevante grænseflader)

Javas hændelseshåndtering

1. Når der indtræffer en hændelse, bestemmes først kilden og hændelsestypen.

2. Hvis der er registreret en eller flere lyttere hos kilden til denne hændelsestype, skabes et hændelsesobjekt.

3. For enhver af disse lyttere kaldes metoden til hændelseshåndtering med hændelsesobjektet som parameter.

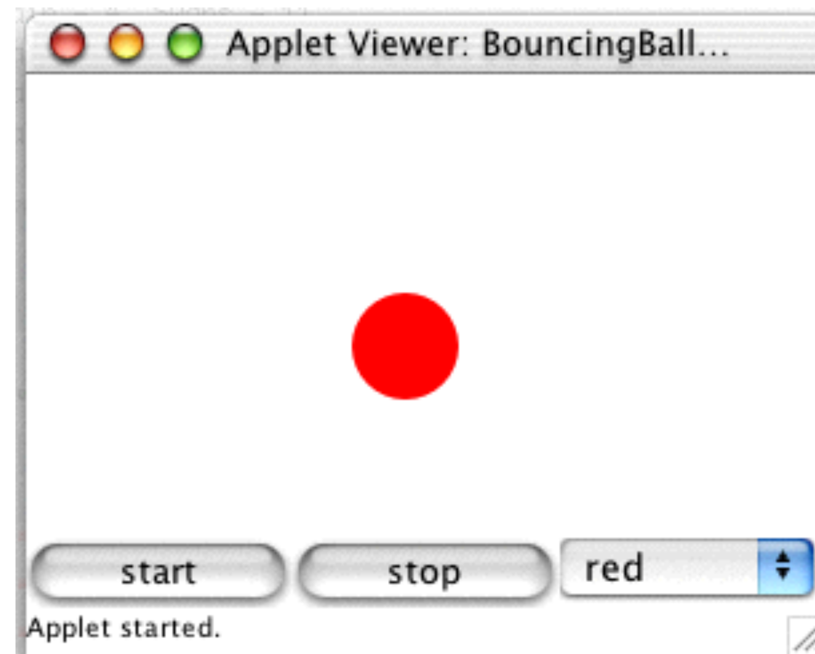
Håndtering af knaptryk og valg

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

public class NestedPanels2 extends NestedPanels
             implements ActionListener, ItemListener {
    public NestedPanels2() {
        super();
        choice.addItemListener(this);
        registerButtonHandler(this);
    }

    public void itemStateChanged(ItemEvent event)  {
        Choice choice = (Choice) event.getSource();
        messageBar.setText("Choice selected: " + choice.getItem());
    }
```

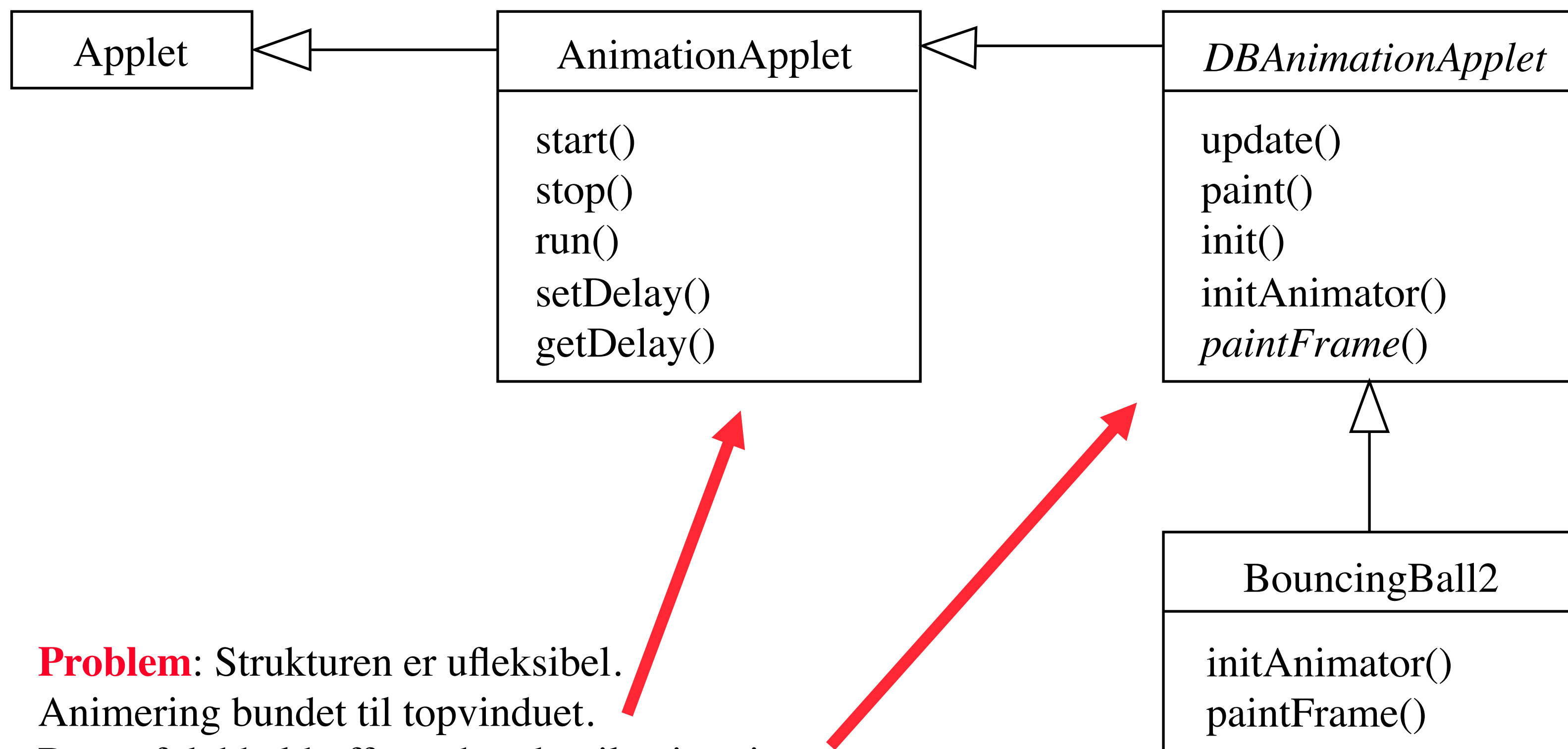
fortsættes

```
    public void actionPerformed(ActionEvent event) {
        Button source = (Button) event.getSource();
        messageBar.setText("Button pushed: " + source.getLabel());
    }

    protected void registerButtonHandler(Component comp) {
        if (comp instanceof Button) {
            Button button = (Button) comp;
            button.addActionListener(this);
        } else if (comp instanceof Container) {
             Container container = (Container) comp;
             int n = container.getComponentCount();
             for (int i = 0; i < n; i++)
                 registerButtonHandler(container.getComponent(i));
        }
    }
}
```

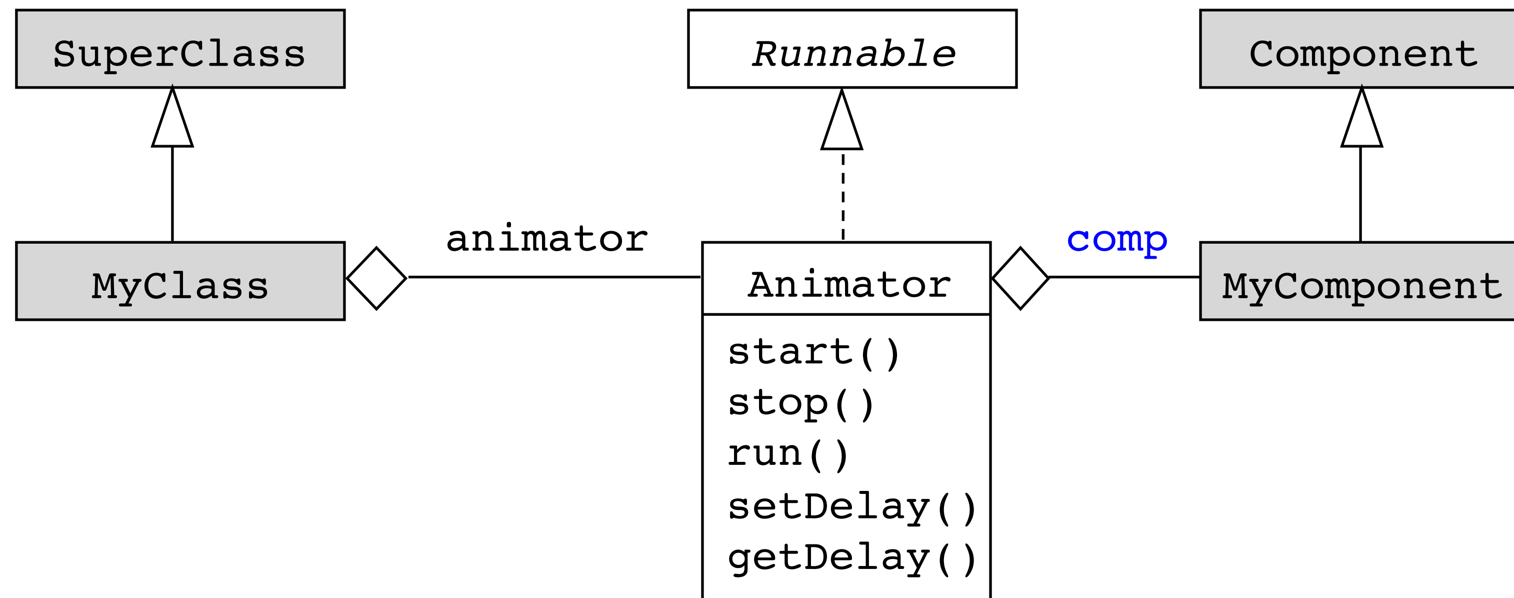
En hoppende bold med kontrol



Tidligere klassediagram

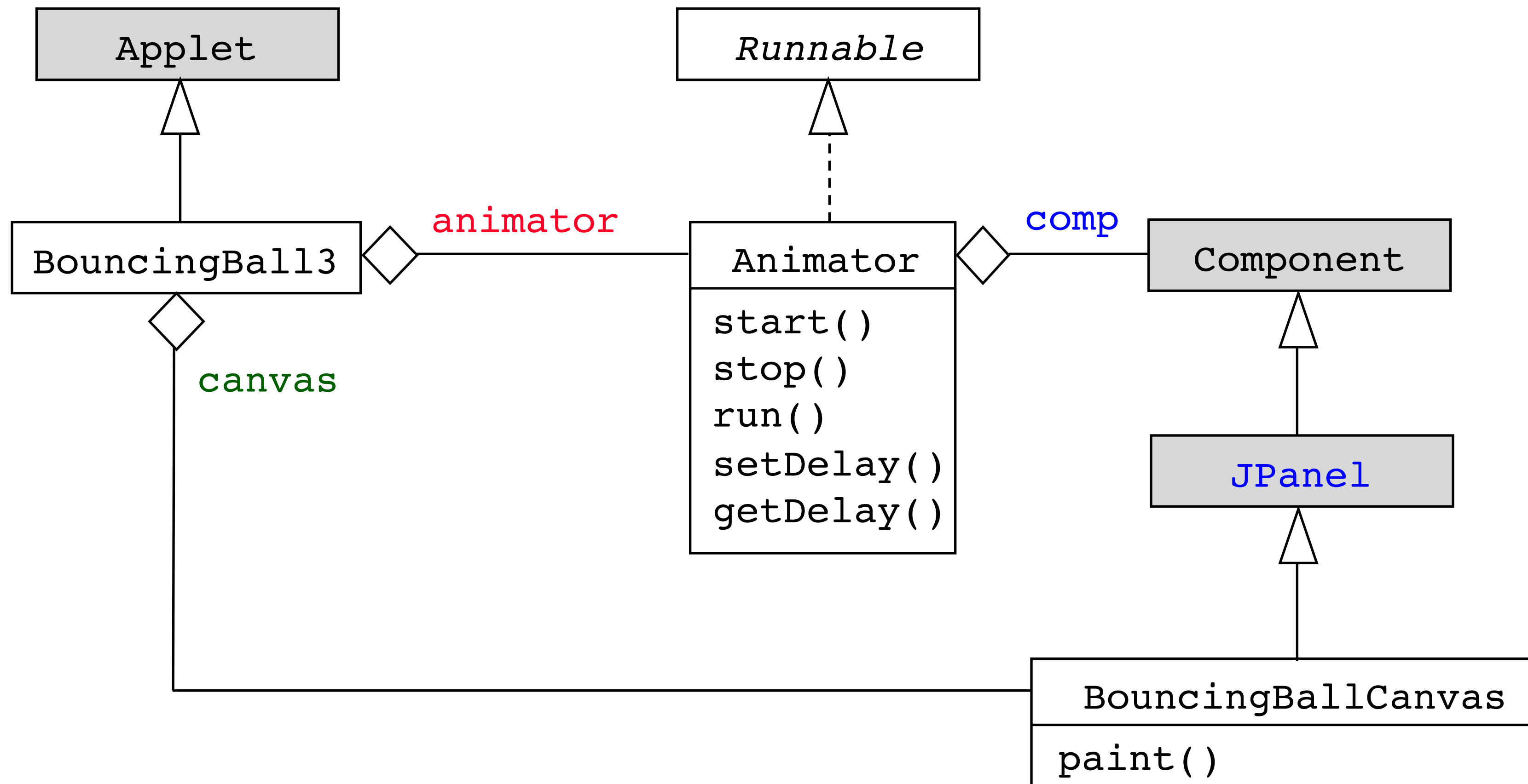


Delegationsbaseret klasse til animering

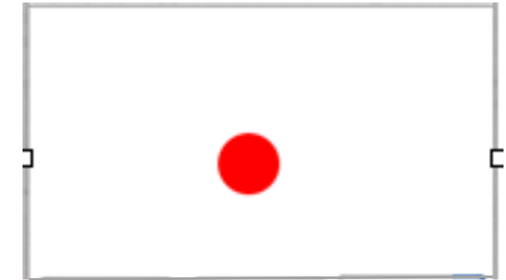


```
public class Animator implements Runnable {
    public Animator(Component comp) {
        this.comp = comp;
    }

    public void start() {
        if (animationThread == null)
            animationThread = new Thread(this);
        animationThread.start();
    }

    public void stop() { animationThread = null; }

    public void run() {
        while (animationThread != null) {
            try {
                Thread.sleep(delay);
            } catch (InterruptedException e){}
            comp.repaint();
        }
    }

    final public void setDelay(int delay) { this.delay = delay; }
    final public int getDelay() { return delay; }

    protected Component comp;
    protected int delay = 100;
    protected Thread animationThread;
}
```

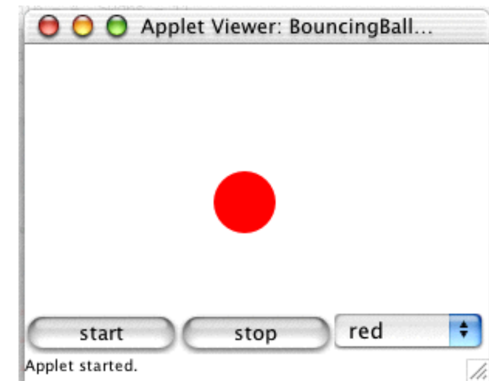
BouncingBall3



```
public class BouncingBallCanvas extends JPanel {
    public BouncingBallCanvas() {
        super(true); // double-buffered
    }

    public void initCanvas() {
        d = getSize();
        x = d.width * 2 / 3 ;
        y = d.height - radius;
    }

    public void paint(Graphics g) { ... }

    public void setBallColor(Color c) { ballcolor = c; }

    protected int x, y, dx = -2, dy = -4, radius = 20;
    protected Color ballcolor = Color.red;
    protected Dimension d;
}
```



```
public class BouncingBall3 extends java.applet.Applet {
  public BouncingBall3() {
    setLayout(new BorderLayout());
    canvas = new BouncingBallCanvas();
    add(canvas, BorderLayout.CENTER);
    animator = new Animator(canvas);
    controlPanel = new JPanel();
    controlPanel.setLayout(new GridLayout(1, 0));
    JButton startButton = new JButton("start");
    controlPanel.add(startButton);
    JButton stopButton = new JButton("stop");
    controlPanel.add(stopButton);
    Choice choice = new Choice();
    choice.addItem("red");
    choice.addItem("green");
    choice.addItem("blue");
    controlPanel.add(choice);
    add(controlPanel, BorderLayout.SOUTH);
    startButton.addActionListener(new ButtonHandler(new StartCommand()));
    stopButton.addActionListener(new ButtonHandler(new StopCommand()));
    choice.addItemListener(new ColorChoiceHandler());
  }
  // ...
```



```
public void init() {
    String att = getParameter("delay");
    if (att != null) {
        int delay = Integer.parseInt(att);
        animator.setDelay(delay);
    }
    validate();  // Layout components again. Necessary!
    canvas.initCanvas();
}

public void start() {
    animator.start();
}

public void stop() {
    animator.stop();
}

protected BouncingBallCanvas canvas;
protected Animator animator;
protected Panel controlPanel;
```

red

```
protected class ColorChoiceHandler implements ItemListener {
    public void itemStateChanged(ItemEvent event)  {
        if ("red".equals(event.getItem()))
            canvas.setBallColor(Color.RED);
        else if ("green".equals(event.getItem()))
            canvas.setBallColor(Color.GREEN);
        else if ("blue".equals(event.getItem()))
            canvas.setBallColor(Color.BLUE);
        canvas.repaint();
    }
}
```

```java
public interface Command {
    void execute();
}

protected class StartCommand implements Command {
    public void execute() {
        start();
    }
}

protected class StopCommand implements Command {
    public void execute() {
        stop();
    }
}
```

```
protected class ButtonHandler implements ActionListener {
    private Command cmd;

    public ButtonHandler(Command cmd) {
        this.cmd = cmd;
    }

    public void actionPerformed(ActionEvent event) {
        if (cmd != null)
            cmd.execute();
    }
}
```

Designmønsteret Command

Kategori:

Adfærdsmæssigt designmønster

Hensigt:

At indkapsle en handling i et objekt, således at handlinger kan overføres som parametre, sættes i kø og eventuelt trækkes tilbage.

Anvendelse:

- Når handlinger skal overføres som parametre.
- Når handlinger skal sættes i kø og udføres senere.
- Når handlinger skal kunne trækkes tilbage.

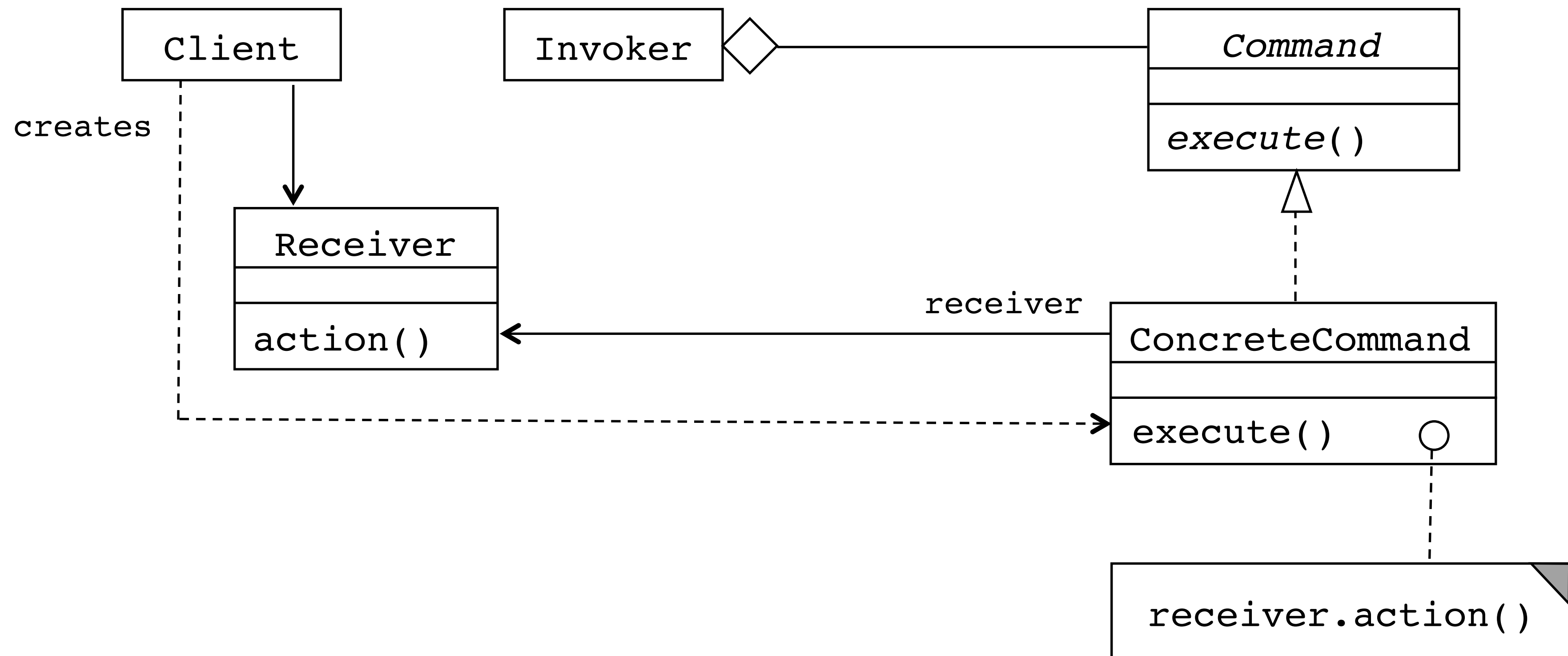
Andre navne:

Action, Transaction

Designmønsteret Command

(fortsat)

Struktur:



Designmønsteret Command
(fortsat)

Deltagere:

Command (f.eks. `Command`), der definerer en grænseflade til udførelse af handlinger eller tilbagetrækning af udførte handlinger.

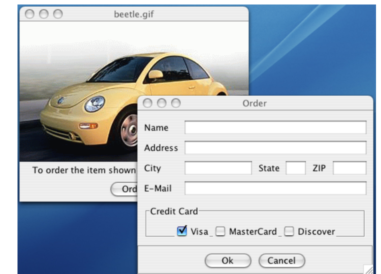
Receiver (f.eks. `BouncingBall3`), der ved, hvorledes handlinger skal udføres.

ConcreteCommand (f.eks. `StartCommand`), der delegerer udførelsen af en handling videre til *Receiver*.

Client (f.eks. `BouncingBall3`), som skaber de konkrete kommandoer og binder dem til deres modtagere.

Invoker (f.eks. `ButtonHandler`), som beder kommandoer om at udføre deres tilknyttede handlinger.

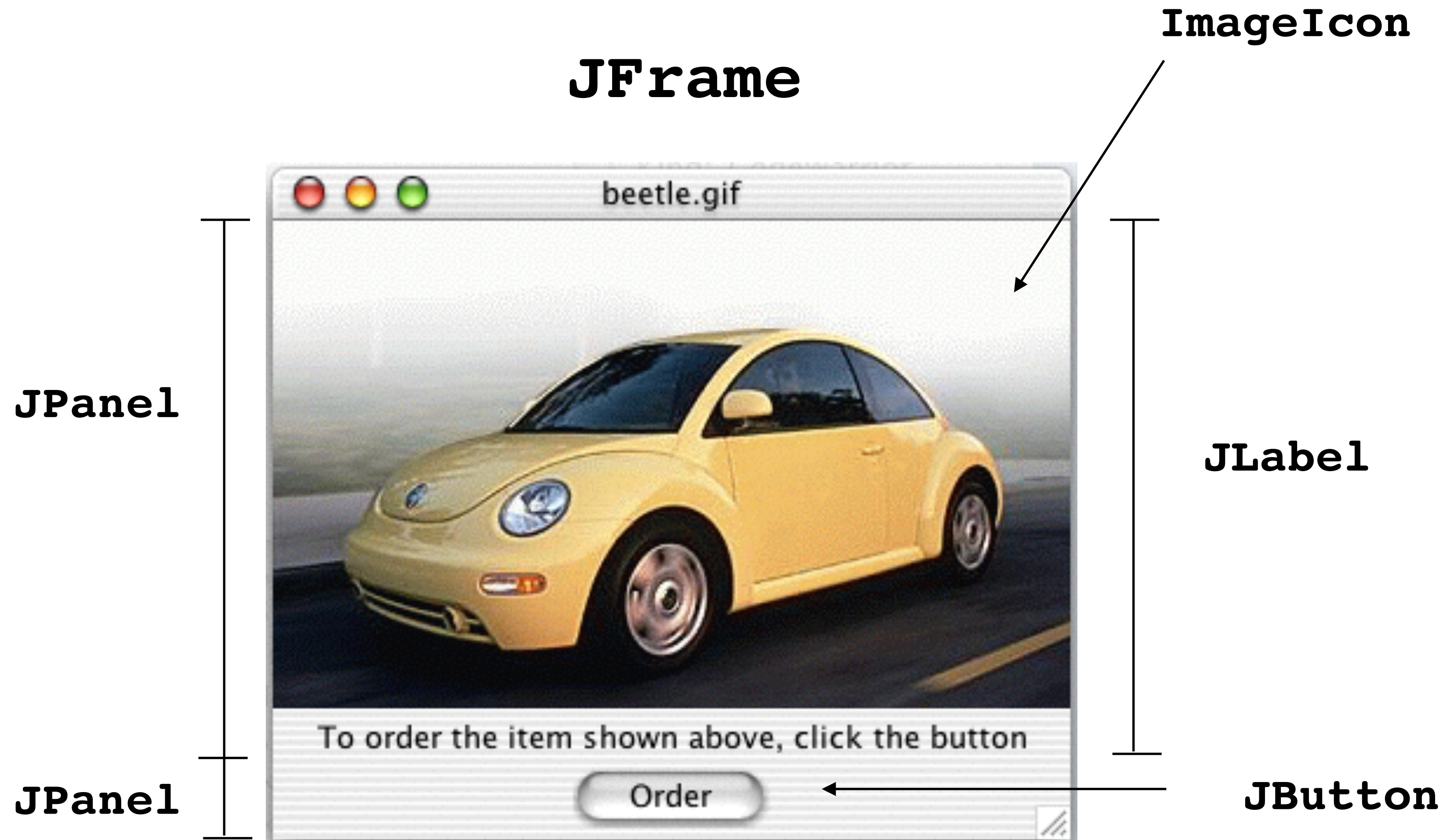
Rammer og dialoger



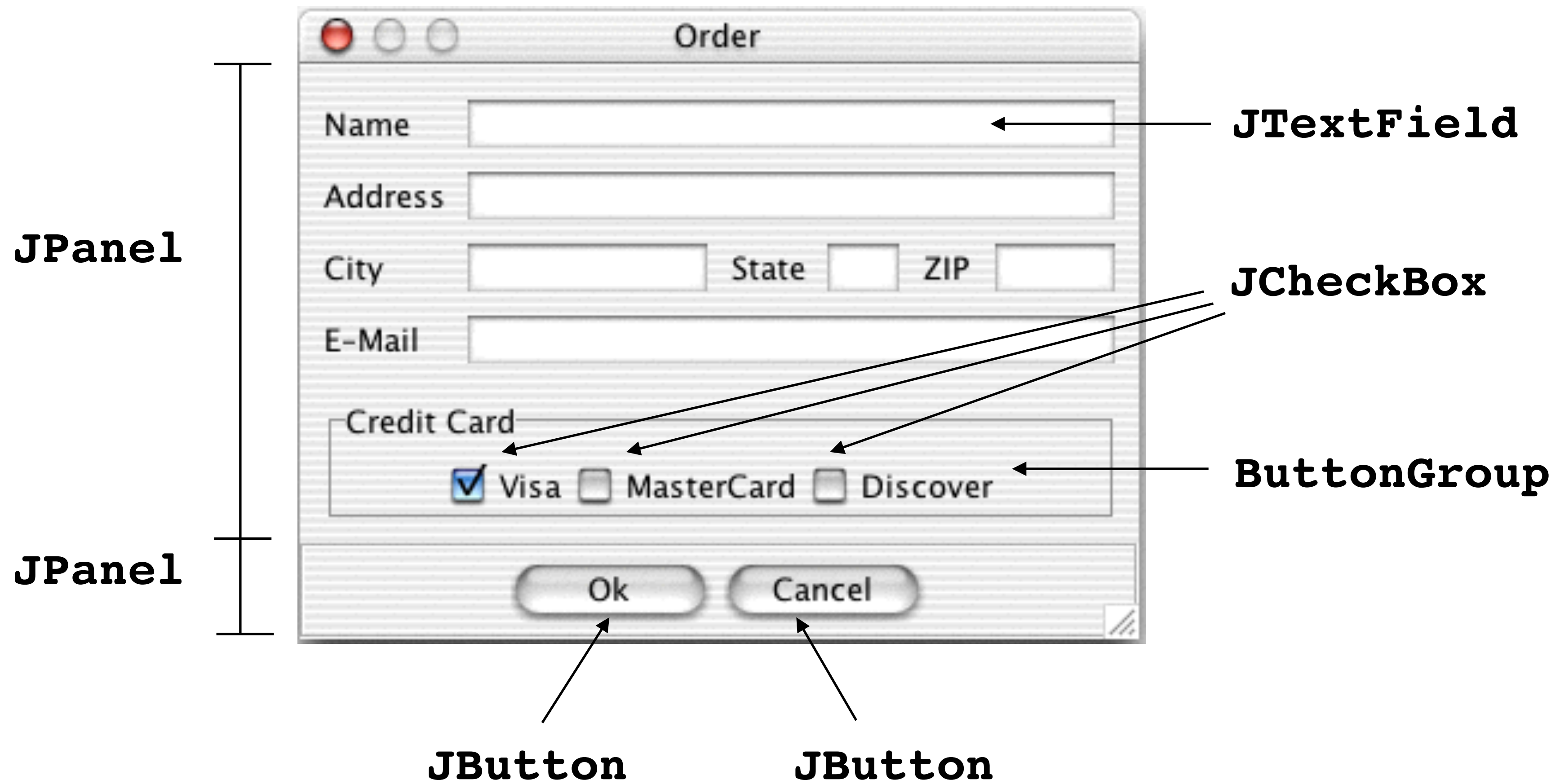
En *ramme* (frame) er et topniveau-vindue med en titel, en rand og kontrolknapper i rammens øverste hjørner.

En *dialog* er et popup-vindue, der sædvanligvis kræver brugerens umiddelbare opmærksomhed. Dialoger bruges til at vise meddelelser og modtage input.

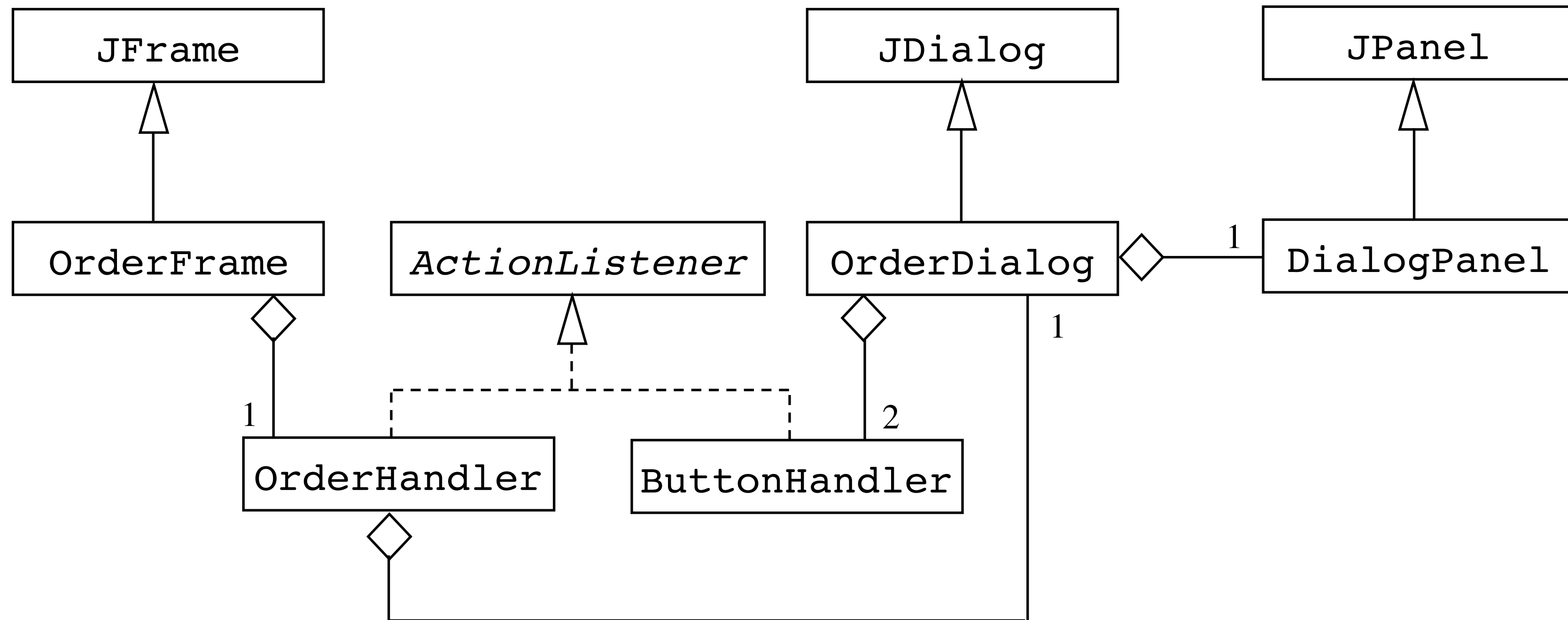
GUI til E-handel



JDialog



UML-diagram



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

public class OrderFrame extends JFrame {
    public OrderFrame(String imageFile) {
        setTitle(imageFile);
        getContentPane().setLayout(new BorderLayout());
        Icon image = new ImageIcon(imageFile);
        JLabel center = new JLabel(
            "To order the item shown above, click the button",
            image, SwingConstants.CENTER);
        center.setHorizontalTextPosition(SwingConstants.CENTER);
        center.setVerticalTextPosition(SwingConstants.BOTTOM);
        JPanel bottom = new JPanel();
        JButton orderButton = new JButton("Order");
        bottom.add(orderButton);
        orderButton.addActionListener(new OrderHandler());
        getContentPane().add(center, BorderLayout.CENTER);
        getContentPane().add(bottom, BorderLayout.SOUTH);
        addWindowListener(new AppCloser());
        // eller setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);
    }
```

fortsættes

```java
static class AppCloser extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
}

class OrderHandler implements ActionListener {
    JDialog dialog;

    public void actionPerformed(ActionEvent evt) {
        if (dialog == null)
            dialog = new OrderDialog(OrderFrame.this);
        dialog.setVisible(true);
    }
}

public static void main(String[] args) {
    if (args.length > 0)
        new OrderFrame(args[0]);
}
```

```
public class OrderDialog extends JDialog {
    DialogPanel dialogPanel;


    public OrderDialog(JFrame owner) {
        super(owner, true);   // modal dialog
        setTitle("Order");
        JButton okButton = new JButton("Ok");
        JButton cancelButton = new JButton("Cancel");
        ButtonHandler bHandler = new ButtonHandler();
        okButton.addActionListener(bHandler);
        cancelButton.addActionListener(bHandler);
        JPanel bottom = new JPanel();
        bottom.add(okButton);
        bottom.add(cancelButton);
        bottom.setBorder(BorderFactory.createEtchedBorder());
        dialogPanel = new DialogPanel();
        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(dialogPanel, BorderLayout.CENTER);
        getContentPane().add(bottom, BorderLayout.SOUTH);
    }
```

fortsættes

```
class ButtonHandler implements ActionListener {
    public void actionPerformed(ActionEvent evt) {
        JButton button = (JButton) evt.getSource();
        String label = button.getText();
        if (label.equals("Ok")) {
            System.out.println("An order is received:");
            System.out.println("\tName:    " +
                dialogPanel.nameField.getText());
            System.out.println("\tAddress:    " +
                dialogPanel.addressField.getText());
            // ...
            if (dialogPanel.visaBox.isSelected())
                System.out.println("Visa");
            else if (dialogPanel.mcBox.isSelected())
                // ...
        }
        dialogPanel.reset();
        setVisible(false);
    }
}
```

```
class DialogPanel extends JPanel {
    JLabel nameLabel, addressLabel, cityLabel,
           stateLabel, zipLabel, emailLabel;
    JTextField nameField, addressField, cityField,
               stateField, zipField, emailField;
    JCheckBox visaBox, mcBox, discoverBox;
    JPanel creditCard;
    ButtonGroup group;

    DialogPanel() { ... }
    public void doLayout() { ... }
    public void reset() { ... }

    public Dimension getPreferredSize() {
        return new Dimension(350, 200);
    }

    public Dimension getMinimumSize() {
        return new Dimension(350, 200);
    }
}
```

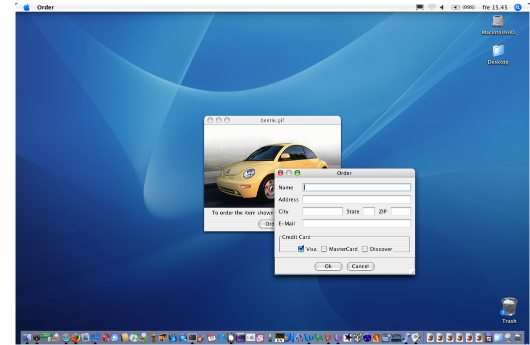
```
DialogPanel() {
    nameLabel = new JLabel("Name");
    nameField = new JTextField();
    // ...
    creditCard = new JPanel();
    visaBox = new JCheckBox("Visa", true);
    mcBox = new JCheckBox("MasterCard");
    discoverBox = new JCheckBox("Discover");
    creditCard.add(visaBox);
    creditCard.add(mcBox);
    creditCard.add(discoverBox);
    creditCard.setBorder(
        BorderFactory.createTitledBorder("Credit Card"));
    group = new ButtonGroup();
    group.add(visaBox); group.add(mcBox); group.add(discoverBox);
    add(nameLabel);
    add(nameField);
    // ...
    add(creditCard);
}
```

JDialog

```
public void doLayout() {
    nameLabel.setBounds(10, 10, 60, 30);
    nameField.setBounds(70, 15, 270, 20);
    addressLabel.setBounds(10, 40, 60, 30);
    addressField.setBounds(70, 45, 270, 20);
    cityLabel.setBounds(10, 70, 60, 30);
    cityField.setBounds(70, 75, 100, 20);
    stateLabel.setBounds(180, 70, 40, 30);
    stateField.setBounds(220, 75, 30, 20);
    zipLabel.setBounds(260, 70, 30, 30);
    zipField.setBounds(290, 75, 50, 20);
    emailLabel.setBounds(10, 100, 60, 30);
    emailField.setBounds(70, 105, 270, 20);
    creditCard.setBounds(10, 140, 330, 50);
}
```

```
public void reset() {
    nameField.setText("");
    addressField.setText("");
    cityField.setText("");
    stateField.setText("");
    zipField.setText("");
    emailField.setText("");
    visaBox.setSelected(true);
}
```

Placering af ramme og dialog på skærm




```
public OrderFrame() {
    ...
    Dimension dim = getToolkit().getScreenSize();
    Rectangle bounds = getBounds();
    setLocation((dim.width - bounds.width) / 2,
                (dim.height - bounds.height) / 2);
}
```

```
public OrderDialog(JFrame owner) {
    ...
    Rectangle bounds = getBounds();
    setLocation(owner.getX() + bounds.width / 2,
                owner.getY() + bounds.height / 2);
}
```

Terminering af program ved lukning af vindue

I `Frame` eller `JFrame` indsættes

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

Eller: i `JFrame` indsættes

```
setDefaultCloseOperation(EXIT_ON_CLOSE);
```

Lyttere som anonyme indre klasser

```
JButton okButton = new JButton("OK");

okButton.addActionListener(
    new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            System.out.println("An order is received:");
            ...
        }
    }
);
```

Ugeseddel 6

5. oktober - 12. oktober

- Læs kapitel 9 i lærebogen (side 397 - 463)

- Løs **projekt** 8.2. Udvid eventuelt også appletten, så det bliver muligt at tilføje en ny bold på det sted, hvor der klikkes med musen.
  Løsningen på opgave 7.5 kan hentes fra kursets hjemmeside og benyttes som udgangspunkt.

- Løs opgaven på næste side.

Ekstraopgave 4

Skriv en applet, der stiller brugeren en opgave i det såkaldte 15-spil.

Givet en udgangsstilling med 15 nummererede brikker som den vist nedenfor

8	9		15
6	11	12	1
3	5	4	7
2	10	14	13

skal brugeren flytte brikkerne, så følgende stilling opstår:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

I hvert træk flyttes en af nabobrikkerne til det tomme felt. Kun det tomme felts nabobrikker i vandret og lodret retning må flyttes i et træk.