Videregående programmering i Java

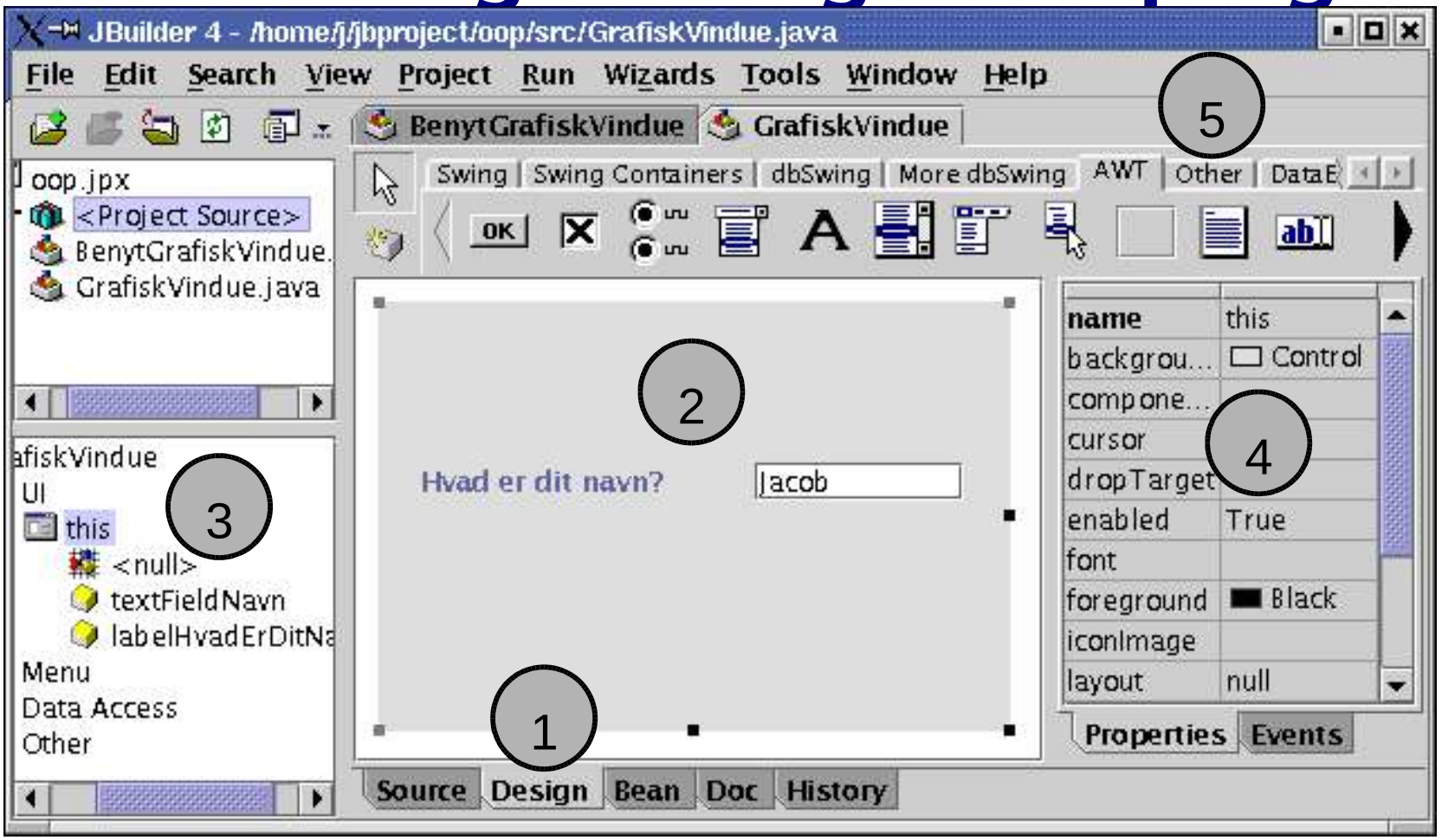
Dag 6 – Komponenter (og lidt Swing og MVC)

Læsning: VP 4, evt. VP 6

Grafiske komponenter

```
import java.awt.*;
public class GrafiskVindue extends Frame
{
  Label labelHvadErDitNavn = new Label();
  TextField textFieldNavn = new TextField();

  public GrafiskVindue() {
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  private void jbInit() throws Exception {
    labelHvadErDitNavn.setText("Hvad er dit navn?");
    labelHvadErDitNavn.setBounds(new Rectangle(15, 69, 108, 15));
    textFieldNavn.setText("Jacob");
    textFieldNavn.setBounds(new Rectangle(141, 61, 112, 29));
    this.setLayout(null);
    this.add(textFieldNavn, null);
    this.add(labelHvadErDitNavn, null);
  }

  public void paint(Graphics g)
  {
    g.drawLine(0,0,50,50);
    g.fillOval(5,20,300,30);
    g.setColor(Color.green);

    String navn = textFieldNavn.getText();
    for (int i=0; i<50; i=i+10)
      g.drawString("Hej "+navn+" !",100+i,30+i);

  }
}
```

Visuelt design af et grafisk program



(1) Design-fanen
(2) Visuelt designområde
(3) Struktur af programmet
(4) Egenskaber (og hændelser) på valgt element
(5) Komponentfaner

Karakteristika ved javabønner

Bønner *skal* have en konstruktør uden parametre

Udviklingsværktøjet kan oprette komponenter på en ensartet måde:

```
Rystetekst rystetekst1 = new Rystetekst();
```

Bønner kan have egenskaber

Udviklingsværktøjet kan generere koden, der sætter egenskaber.
Egenskaben *tekst* understøttes ved at bønnen definerer metoderne:

```
public void setTekst(String t)
public String getTekst()
```

Bønner bør defineres i en pakke

Bønner bør være afgrænsede og uafhængige

Bønner kan understøtte hændelses-lyttere

Udviklingsværktøjet kan generere koden, der registrerer lyttere.

Bønnen kan understøtte Action-hændelser ved at definere metoderne:

```
public void addActionListener(ActionListener l)
public void removeActionListener(ActionListener l)
```

Bønner kan have tilknyttet ekstra information

Denne information er *kun* til hjælp for udviklingsværktøjet:

- Hvilket ikon bønnen skal repræsenteres af i værktøjet
- Hvilke egenskaber der findes, beskrivelser og hvordan den aflæses/sættes
- Hvordan de redigeres. Der kan tilknyttes en klasse, der bestemmer f.eks.:
  - Hvilke værdier der er mulige
  - Hvilken javakode der skal sættes ind i kildeteksten
  - Om et skræddersyet redigeringsvindue skal dukke op

Genbrugelige komponenter

- Komponenter er programmørens byggeklodser
  - De kan bruges igen og igen i mange sammenhænge
  - De kan sættes sammen på alle mulige måder
  - De er nemme at indstille
  - Udviklingsværktøj kan generere programkoden for programmøren
- Grafiske brugergrænseflader er som regel helt bygget op af komponenter!
- Komponenter i Java hedder Javabønner (eng.: JavaBeans)

Komponentbaseret udvikling

Eksempel: Bønnen TextField

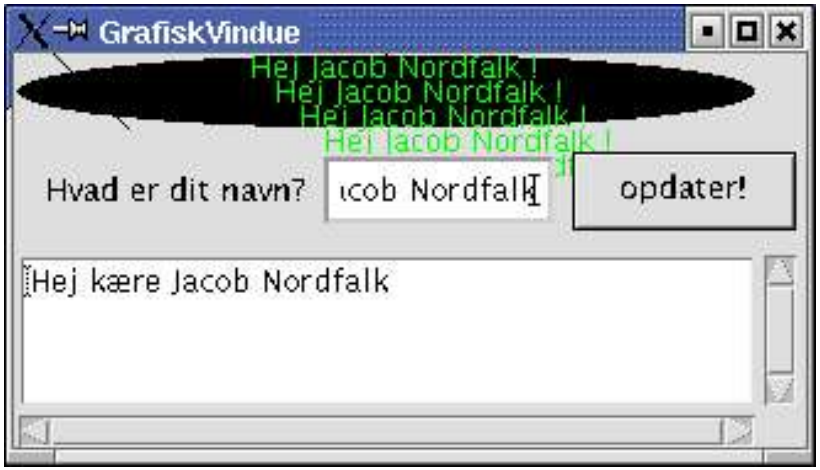
Egenskab	*Type*	*Egenskab sættes med*	*Egenskab aflæses med*
text	String	setText(String t)	getText()
editable	boolean	setEditable(boolean rediger)	isEditable()
columns	int	setColumns(int bredde)	getColumns()
echoChar	char	setEchoChar(char tegn)	getEchoChar()

Bruge en javabønne fra et udviklingsværktøj

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

public class BenytBoenneMedVaerktoej extends Frame
{
  TextField textFieldNavn = new TextField();      // opret bønnen

  public BenytBoenneMedVaerktoej() {
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  private void jbInit() throws Exception {
    textFieldNavn.setText("Jacob");               // sæt egenskaben text
    textFieldNavn.setBounds(new Rectangle(141, 61, 112, 29));
    this.setLayout(null);
    this.add(textFieldNavn, null);
  }
}
```

Hændelser - genereret af værktøj

```
import java.awt.*;
public class GrafiskVindue extends Frame
{
  Button buttonOpdater = new Button();
  TextArea textAreaHilsen = new TextArea();
  ...

  private void jbInit() throws Exception {
    ...

    buttonOpdater.setLabel("opdater!");
    buttonOpdater.setBounds(new Rectangle(231, 60, 91, 32));
    buttonOpdater.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        buttonOpdater_actionPerformed(e);
      }
    });
    textAreaHilsen.setText("Her kommer en tekst...");
    textAreaHilsen.setBounds(new Rectangle(6, 102, 316, 78));
    this.add(textAreaHilsen, null);
    this.add(buttonOpdater, null);
  }

  void buttonOpdater_actionPerformed(ActionEvent e) {
    String navn = textFieldNavn.getText();
    System.out.println("Opdater! navn="+navn);
    textAreaHilsen.setText("Hej kære "+navn);
    repaint(); // gentegn vinduet, så paint() bliver kaldt
  }

  ...
}
```



Designmønstret Observatør/Lytter

(eng.: Observer/Listener)
eller Abonnent (eng.: Publisher-Subscriber)

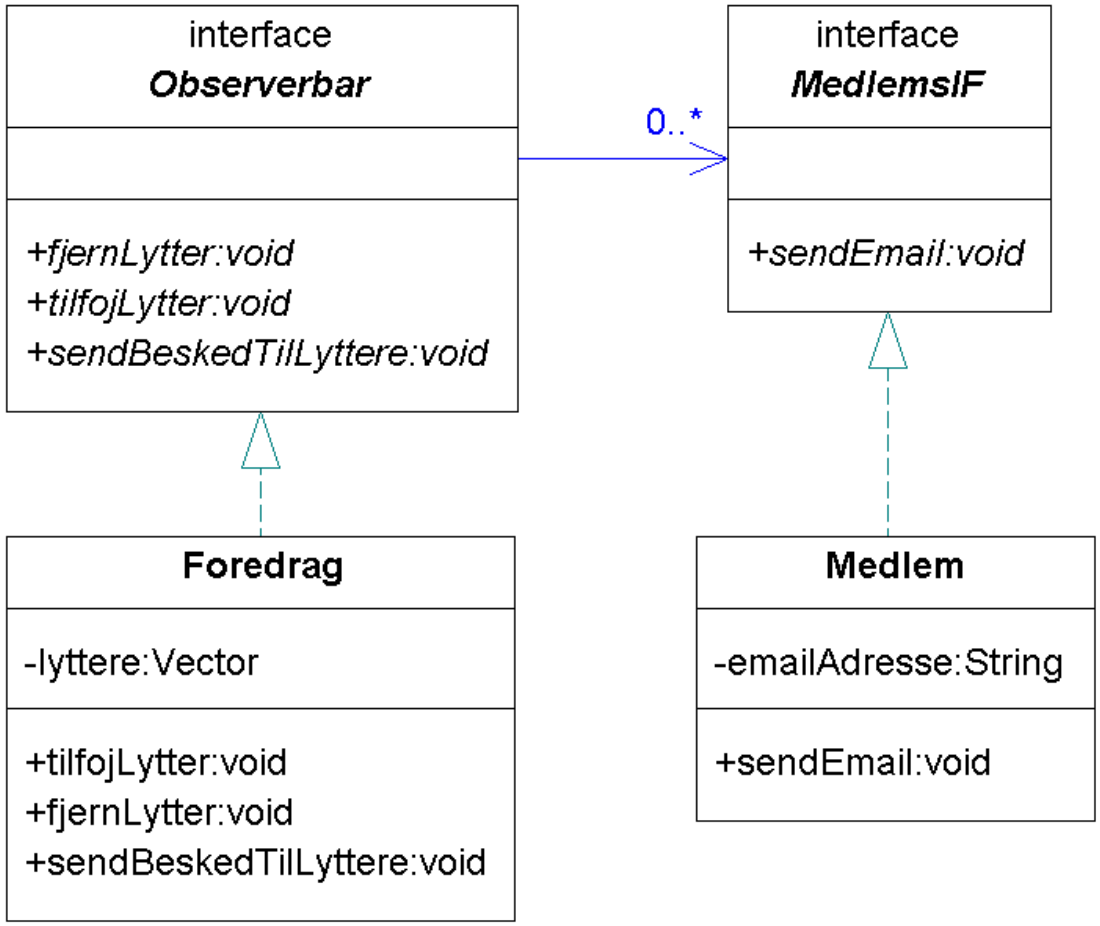
('abonnere' på, at en ting (hændelse) sker)

Problem: Et objekt skal kunne underrette nogle andre objekter om en eller anden ændring eller hændelse, men det er ikke hensigtsmæssigt, at objektet kender direkte til de andre objekter.

Løsning: Lad lytterne (observatørerne) implementere et fælles interface (eller arve fra en fælles superklasse) og registrere sig hos det observable (observerbare) objekt. Det observable objekt kan herefter underrette lytterne gennem interfacet, når der er brug for det.

Designmønstret Observatør/Lytter

- Eksempel: Fælles kalender for en forening, der udbyder foredrag
  - Hvis der sker ændringer i planen, skal kun de interesserede medlemmer have besked, dvs. de, som har tilmeldt sig et givent foredrag.
- xxx Interface Observerbar slettes, MedlemsIF omdøbes til Observatoe



Designmønstret Observatør/Lytter

- Programkode til observabelt objekt (og javabønne)

```
public class MinButton extends Component
{
  /** Lyttere til denne bønne */
  private ArrayList lyttere = new ArrayList(2);

  public void addActionListener(ActionListener l) { ly

  public void removeActionListener(ActionListener l) {

  /** Sender en hændelse til lyttere. Lytterne er de,
   *  tilføjet med kald til addActionListener() */
  protected void underretLytterne(ActionEvent hændelse
  {
    for (Iterator i=lyttere.iterator(); i.hasNext();
    {
      ActionListener l = (ActionListener) i.next();
      l.actionPerformed(hændelse);
    }
  }

  public void enMetode()
  {
    if (...) {
      // opret en Action-hændelse, der kommer fra denne komponent og send den
      // brug konstruktøren new ActionEvent( afsenderobjekt, id, beskrivelse)
      ActionEvent hændelse = new ActionEvent(this, 0, "der skete noget!");
      underretLytterne(hændelse);
    }
  }
  ...
}
```

Værktøjets kode:

```
import java.awt.*;
public class GrafiskVindue extends Frame
{
  MinButton buttonOpdater = new MinButton();
  ...

  private void jbInit() throws Exception {
    ...
    buttonOpdater.setLabel("opdater!");
    buttonOpdater.addActionListener(new java.awt.e
      public void actionPerformed(ActionEvent e) {
        buttonOpdater_actionPerformed(e);
      }
    });
    textAreaHilsen.setText("Her kommer en tekst...
    ...
  }

  void buttonOpdater_actionPerformed(ActionEvent e
    System.out.println("Knappen blev trykket!");
  }

  ...
}
```