

Grundlæggende Programmering, efterår 2000

Forelæsning 9, onsdag 1. november 2000

Øversigt

- Klasse-grænseflader ('interfaces')
- Grafiske brugergrænseflader
- Komponenter: Knapper, tekstfelter, checkboks...
- Vinduer (containere)
- Hændelser, hændelsesstyret programmering
- Indre klasser

Klasse-grænseflader ('interfaces'): en erstatning for multipel nedarvning

En *klasse-grænseflade* (interface) kan opfattes som en ultra-abstrakt klasse:

- En klasse-grænseflade kan kun erklære felter der er konstante.
- Derfor er nøgleordet `final` underforstået: feltene *skal* være konstante (men det er OK at skrive `final`).
- En klasse-grænseflade kan kun erklære metoder der er abstrakte og synlige overalt. Derfor er nøgleordene `abstract` og `public` underforståede (men må gerne skrives).
- En metode beskrevet i en grænseflade skal være `public` i klasser der implementerer grænsefladen.
- En klasse-grænseflade kan ikke erklære konstruktører.

En klasse kan *implementere* vilkårlig mange grænseflader (og stadig *udvide* en superklasse).

For at implementere en klasse-grænseflade skal klassen implementere alle klasse-grænsefladens metoder.

En klasse-grænseflade (interface) kan udvide andre klasse-grænseflader (med `extends`).

Enkel og multipel nedarvning

I Java kan en klasse kun have én umiddelbar superklasse. I visse andre sprog (C++) kan man have flere.

Dette kaldes *multipel nedarvning* ('multiple inheritance').

Multipel nedarvning er nyttig når man arbejder med to forskellige begrebshierarkier på en gang.

F.eks. beholdere (Vessel, Tank, Barrel) og farver (Plain, Colored).

- En farvet tønde er både farvet (Colored) og en beholder (Vessel). Så et objekt skal kunne være Colored og Vessel på en gang.
- En almindelig tønde er en Vessel, men ikke Colored.
- Så Vessel kan ikke være en subklasse af Colored.
- Et farvet papirark er Colored, men ikke en Vessel.
- Så Colored kan ikke være en subklasse af Vessel.

Java har enkel nedarvning ('single inheritance'): en klasse kan kun have én umiddelbar superklasse.

Det er fordi multipel nedarvning fører til teoretiske og praktiske problemer.

(Eksempel: I hvilken rækkefølge skal superklassernes konstruktører kaldes?)

(Eksempel: Hvis to metoder med samme signatur arves fra to forskellige superklasser, hvilken en skal så bruges?)

Eksempel: Objekter der både er beholdere og farvede (Vessel11.java)

```
interface Colored {
    Color getColor();
}

class ColoredBarrel extends Barrel implements Colored {
    private Color c;

    ColoredBarrel(double radius, double height, Color c)
    { super(radius, height); this.c = c; }

    public Color getColor()
    { return c; }
}

public class Vessel16 {
    static void printColor(Colored cobj)
    { System.out.println(cobj.getColor().toString()); }

    public static void main(String[] args) {
        Vessel v1 = new Tank(15, 9, 12);
        ColoredBarrel cb = new ColoredBarrel(2.5, 8, Color.red);
        printColor(cb);
    }
}
```

Grafiske brugergrænseflader (GUI)

I 'gammeldags' programmer stillede programmet en række spørgsmål man skulle svare på.

Programmet styrede dialogen: i hvilken rækkefølge skal brugeren indtaste data osv.

Eksempel: Programmet Valuta

Moderne programmer har *grafiske brugergrænseflader*.

Brugeren styrer selv dialogen ved at udfylde tekstfelter, trykke på knapper, vælge fra menuer, osv.

Grafisk brugergrænseflade = 'graphical user interface' = GUI.

Java har standard-klasser til at konstruere grafiske brugergrænseflader.

De er samlet i Javas *Abstract Window Toolkit* (AWT).

Komponenter

Den synlige del af en GUI er opbygget af *komponenter*, dvs. objekter af klassen Component.

Component har en række subklasser der svarer til forskellige slags komponenter, for eksempel:

```
Component <- Button           (knap)
               <- Canvas       (lærred)
               <- Choice        (valgmenu)
               <- Checkbox      (checkboxoks)
               <- Label         (etiket)
               <- List          (liste)
               <- Scrollbar     (skyder)
               <- TextArea      (tekstområde)
               <- TextField     (tekstfelt)
```

De forskellige slags komponenter er beslægtede, og derfor er de organiseret i et klassehierarki:

Enhver komponent har en placering og en størrelse.

En knap er en komponent. Desuden har den en påskrift og kan reagere på at man trykker på den.

En tekstkomponent er en komponent. Desuden indeholder den en aktuell tekst.

Et tekstfelt er en tekstkomponent. Den har plads til én linje tekst.

Et tekstområde er en tekstkomponent. Det har plads til flere tekstlinjer, og det har skydere ('scrollbars').

Hvad skal jeg inkludere?

Enhver Java-fil der bruger AWT skal importere disse pakker:

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

Information om AWT

Java AWT er beskrevet i

- L&L: appendix M side 617
- på kursets hjemmeside 'Materialer på nettet'

Der findes masser af smarte værktøjer til konstruktion af GUI'er.

Hér handler det om at *forså hvordan* en GUI virker, så vi laver det fra bunden af.

Vi kan lave grafiske brugergrænseflader med både appletter og almindelige programmer.

Eksemplerne fokuserer almindelige programmer, men appletter virker på næsten samme måde.

Vinduer

Komponenter skal puttes ind i *containere* for at kunne vises på skærmen.

Der er forskellige slags containere. De er alle sammen subklasser af klassen Container:

```
Container <- Panel           <- Applet
               <- ScrollPane
               <- Window      <- Frame
                               <- Dialog <- FileDialog
```

De forskellige slags containere er beslægtede, og derfor er de organiseret i et klassehierarki.

Klassen Frame bruges til at lave et almindeligt vindue.

En container er selv en subklasse af Component (så en container kan indeholde andre containere).

Et meget simpelt GUI-program (FirstGUI.java)

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

class FirstGUI {
    public static void main(String [] args) {
        Frame f = new Frame("Min frame");
        f.setLayout(new FlowLayout());
        Button b = new Button("Min knap");
        f.add(b);
        f.pack();
        f.show();
    }
}
```

Knappen kan trykkes, men der sker ingenting. Senere ser vi hvordan man får knappen til at reagere...

Kaldet `f.setLayout(new FlowLayout())` sætter en layout-manager for `f`; mere om det senere.

En af hver slags komponent (EnAFhver)

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

class EnAFhver {
    public static void main(String args[]) {
        Frame f = new Frame("En af hver");
        f.setLayout(new FlowLayout());
        f.add(new Button("Button"));
        f.add(new Checkbox("Checkbox"));
        f.add(new Label("Label"));
        f.add(new Scrollbar(Scrollbar.HORIZONTAL, 0, 4, 0, 100));
        f.add(new TextField(10));
        f.add(new TextArea(5, 20));
        f.pack();
        f.show();
    }
}
```

Hvordan får vi komponenterne til at reagere på tastatur og mus?

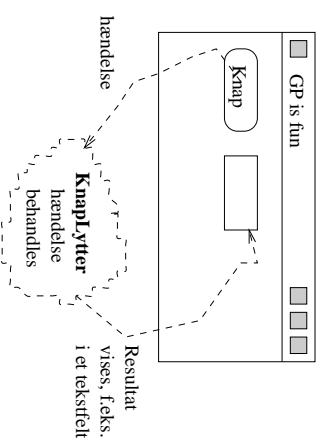
Hændelser (events)

Når brugeren trykker på en knap eller lignende sker der en *hændelse*.

Et GUI-program skal lytte efter hændelser for at kunne reagere på dem.

Programmet styres så af de hændelser brugeren genererer.

I et *hændelses-styret* (event-driven) program udføres programdelene i den rækkefølge brugeren aktiverer dem.



At lytte på hændelser der vedrører en knap (TrykKnap.java)

```
class KnapLytter implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Tak!");
    }
}

class TrykKnap {
    public static void main(String args[]) {
        Frame f = new Frame("Tryk på knappen");
        f.setLayout(new FlowLayout());
        Button b = new Button("Tryk her");
        b.addActionListener(new KnapLytter());
        f.add(b);
        f.pack();
        f.show();
    }
}
```

Nu virker knappen. (Om lidt kan vi også få klik på krydset til at lukke vinduet.)

Hvad er ActionListener?

ActionListener er en klasse-grænseflade (dvs. Java Interface).

Et objekt af type ActionListener bruges til at lytte efter hændelser (objektet er en 'event listener').

ActionListener er erklæret sådan her:

```
public interface ActionListener extends EventListener {
    public abstract void actionPerformed(ActionEvent e);
}
```

Sådan knytter man en 'event listener' til en komponent på skærmen:

- Man skriver en 'lytterklasse', f.eks. KnapLytter, der implementerer Klasse-grænsefladen ActionListener.
- Lytterklassen skal indeholde metoden actionPerformed.
- Man laver et lytteobjekt ud fra lytterklassen (med new KnapLytter()).
- Man knytter lytteobjektet til komponenten, f.eks. knappen b, med ordren
b.addActionListeners(...);

Hver gang der sker en hændelse vedrørende knappen b, så kaldes metoden actionPerformed i lytter-klassen.

Adaptere

Noogle 'lytter'-grænseflader definerer en masse metoder, for eksempel:

Eksempel:

```
interface WindowListener extends EventListener {
    public abstract void windowActivated(WindowEvent e);
    public abstract void windowClosed(WindowEvent e);
    public abstract void windowClosing(WindowEvent e);
    public abstract void windowDeactivated(WindowEvent e);
    public abstract void windowDeiconified(WindowEvent e);
    public abstract void windowIconified(WindowEvent e);
    public abstract void windowOpened(WindowEvent e);
}
```

For at implementere en Klasse-grænseflade skal man implementere *alle* dens metoder.

Det er træls, for tit skal man kun bruge en enkelt af metoderne, f.eks. windowClosing.

En *adapter*/klasse er en hjælpeklasse som definerer alle Klasse-grænsefladens metoder (med tom metodekrop).

Ved at udvide adapterklassen (i stedet for at implementere Klasse-grænsefladen) kan man nøjes med at overskrive de metoder som man skal bruge.

En nyttig udvidelse af JFrame (ClosableFrame, Java)

Vi laver vores egen udgave af JFrame som lukker når der klikkes på krydset.

Det er en specialudgave af JFrame, og bliver derfor naturligvis en subklasse af JFrame:

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

class ClosableFrame extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        e.getWindow().dispose();
        System.exit(0);
    }
}

class ClosableFrame extends JFrame {
    public ClosableFrame(String name) {
        super(name);
        addWindowListeners(new ClosableFrame());
    }
}
```

Men hvad er en WindowAdapter?

Test af ClosableFrame (LytKnap2, Java)

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

class LytKnap2 {
    public static void main(String args[]) {
        JFrame f = new ClosableFrame("Tryk på knappen");
        f.setLayout(new FlowLayout());

        Button b = new Button("Tryk her");
        b.addActionListener(new KnapLytter());
        f.add(b);

        f.pack();
        f.show();
    }
}
```

Etiketter

Man laver en etiket med en given tekst:

```
Label lab = new Label("tekst");
```

Man laver en etiket med en given tekst og justering

```
Label lab = new Label("tekst", Label.RIGHT);
```

Justering kan være `Label.LEFT`, `Label.CENTER` og `Label.RIGHT`.

Tekstfelter

Man laver et tekstfelt med en given bredde:

```
TextField t = new TextField(10); // 10 synlige tegn
```

Man kan læse indholdet af et tekstfelt med metoden `getText()`:

```
String indhold = t.getText();
```

Man kan sætte indholdet med metoden `setText()`:

```
t.setText("En tekst");
```

Man kan bestemme om feltet må rettes af brugeren:

```
t.setEditable(false);
```

Eksempel: En GUI til valutaomregning (ValutaGUI1.java)

```
class ValutaGUI1 {
    TextField argument = new TextField(10);
    Label largument = new Label("Buro:");
    Label lresultat = new Label("Kroner:");
    TextField resultat = new TextField(10);
    Button b = new Button("Beregn");
    Frame f = new ClosableFrame("Valutaomregning");

    class BeregnLytter implements ActionListener {
        public void actionPerformed(ActionEvent e)
        { int eu = Integer.parseInt(argument.getText());
          resultat.setText(Double.toString(eu * 7.42)); }
    }

    public ValutaGUI1()
    { b.addActionListener(new BeregnLytter());
      resultat.setEditable(false);
      f.setLayout(new FlowLayout());
      f.setSize(100, 100);
      f.add(largument); f.add(argument); f.add(b);
      f.add(lresultat); f.add(resultat);
      f.pack(); f.show(); }

    public static void main(String args[])
    { new ValutaGUI1(); }
}
```

Indre klasser

Læg mærke til at klassen `BeregnLytter` er erklæret *inden* i klassen `ValutaGUI1`.

Det har vi ikke gjort før (ikke med vilje i hvert fald).

`BeregnLytter` er en såkaldt *indre klasse* i `ValutaGUI1`.

Et objekt af en indre klasse har adgang til alle felter og metoder fra den omgivende (ydre) klasse.

Dvs. et `BeregnLytter`-objekt kan bruge felterne `argument` og `resultat`.

Indre klasser er praktiske i GUI-programmering

Ved at erklære `BeregnLytter` som en indre klasse opnår vi:

- Metoden `actionPerformed` i `BeregnLytter` kan bruge felterne `argument` og `resultat`.
- Den aktuelle `BeregnLytter` bliver ikke forvekslet med en `BeregnLytter` der hører til et andet program.

Aktivering af ValutaGUI1

Når vi laver et objekt af klasse `ValutaGUI1` oprettes der et vindue (`Frame`) med tekstfelter, knapper osv.

```
class ValutaGUI1 {
    ... argument, resultat, b, f, BeregnLytter som før ...
    public static void main(String args[])
    { new ValutaGUI1(); }
}
```

Når der oprettes et `ValutaGUI1`-objekt (med `new ValutaGUI1()`) så oprettes der også et `Frame`-objekt. `Frame`-objektet ses som et vindue på skærmen.

Vinduet eksisterer, og venter på hændelser, indtil det nedlægges (med `System.exit(0)` eller lignende).

Programmet 'standser' altså ikke bare fordi `main` er blevet færdig. Det venter på at brugeren afslutter det.

Man kan lave adskillige uafhængige kopier af valuta-omregneren:

```
class DobbeltValutaGUI {
    public static void main(String args[]) {
        new ValutaGUI1();
        new ValutaGUI1();
    }
}
```

Retur ('Enter') i et tekstfelt er også en hændelse

Man kan krytze en ActionListener til et tekstfelt for at lytte efter om brugeren trykker retur i tekstfeltet.

I valutaomregneren kan vi sætte BeregnLytter til at lytte på retur i tekstfeltet argument.t.

Vi kan så undvære 'Beregn' knappen.

Eksemplet TekstValutaGUI: en lytter på et tekstfelt (TekstValutaGUI.java)

```
class TekstValutaGUI {
    TextField argument = new TextField(10);
    Label largument = new Label("Euro:");
    Label lresultat = new Label("Kroner:");
    TextField resultat = new TextField(10);
    Frame f = new ClosableFrame("Valutaomregning");

    class BeregnLytter implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            int eu = Integer.parseInt(argument.getText());
            resultat.setText(Double.toString(eu * 7.42));
        }
    }

    public TekstValutaGUI() {
        argument.addActionListeners(new BeregnLytter());
        resultat.setEditable(false);
        f.setLayout(new FlowLayout());
        f.add(largument); f.add(argument);
        f.add(lresultat); f.add(resultat);
        f.pack(); f.show();
    }

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

Tekstområder

Et TextArea-objekt (tekstområde) kan bruges til at vise flere linjer tekst.

Et TextArea har både højde (antal linjer) og bredde (antal tegn):

```
TextArea t = new TextArea(5, 30);
```

Man kan sætte *hele* indholdet af tekstområdet:

```
t.setText("Linie 1\nLinie 2");
```

Man kan læse *hele* indholdet af tekstområdet:

```
String indhold = t.getText();
```

Man kan *tilføje* tekst til området:

```
t.append("En linie\neller 2");
```

Man kan *indsætte* tekst til området ved en bestemt position (f.eks. position 42):

```
t.insert("En linie\neller 2", 42);
```

Man kan bestemme om tekstområdet må rettes af brugeren:

```
t.setEditable(false);
```

Eksempel: Valutaomregning med 'regnestrimmel' (StrimmelValutaGUI.java)

```
class StrimmelValutaGUI {
    TextField argument = new TextField(10);
    Label largument = new Label("Euro:");
    TextArea resultat = new TextArea(5, 30);
    Frame f = new ClosableFrame("Valutaomregning");

    class BeregnLytter implements ActionListener {
        public void actionPerformed(ActionEvent e)
        { int eu = Integer.parseInt(argument.getText());
          double kr = eu * 7.42;
          resultat.append(eu + " euro er " + kr + " kroner\n"); }
    }

    public StrimmelValutaGUI()
    { f.setLayout(new FlowLayout());
      argument.addActionListeners(new BeregnLytter());
      resultat.setEditable(false);
      f.add(largument); f.add(argument);
      f.add(resultat); f.pack(); f.show(); }

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

Opgave: Hvorledes lader vi den sidste beregning stå øverst på skærmen?

Checkboks

En checkboks (eller afkrydsningsboks) er enten afkrydset eller ikke.

Opretelse af en checkboks med et givent navn:

```
Checkbox c = new Checkbox("navn");
```

Afæs checkboksens værdi (afkrydset = true, ellers false):

```
boolean valgt = c.getState();
```

Sæt checkboksens værdi:

```
c.setState(true);
```

Man kan også samle flere checkboks i en gruppe af checkboks, hvor kun en boks kan være afkrydset af gangen, se klassen `CheckboxGroup`.

Valgmener med Klassen Choice

En valgmenu har en række gensidigt udelukkende indstillinger, som kan vælges på et gammeldags fjernsyn.

Opretelse af en valgmenu:

```
Choice c = new Choice();
```

Indsættelse af valgmuligheder:

```
c.addItem("Danmark");
```

```
c.addItem("Europa");
```

```
c.addItem("Øvrige udlænd");
```

Afæs valgmenuens indstilling (hvilket nummer er valgt):

```
int valgtnr = c.getSelectedIndex();
```

Afæs valgmenuens indstilling (hvilken tekst er valgt):

```
String valgttekst = c.getSelectedItem();
```

Sæt valgmenuens indstilling:

```
c.select(2);
```

Den første valgmulighed har nummer 0.

Man kan godt knytte en lyter til en valgmenu, men det er ikke nødvendigt.

Eksempel: Brevporto til Danmark og Europa (BrevportoGUI.java)

```
class BrevportoGUI {
    Checkbox europa = new Checkbox("Europa");
    Checkbox økonomi = new Checkbox("Økonomi");
    Button b = new Button("Beregn");
    TextField resultat = new TextField(10);
    Frame f = new CiosableFrame("Brevporto Danmark/Europa");

    class ButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent e)
        { double porto;
          if (europa.getState()) porto = 4.50; else porto = 4.00;
          if (økonomi.getState()) porto = porto - 0.25;
          resultat.setText(Double.toString(porto)); }
    }

    public BrevportoGUI()
    { b.addActionListener(new ButtonListener());
      resultat.setEditable(false);
      f.setLayout(new FlowLayout());
      f.add(europa); f.add(økonomi); f.add(b); f.add(resultat);
      f.pack(); f.show(); }

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

Eksempel FlerValutaGUI: brug af valgmenu (FlerValutaGUI.java)

```
class FlerValutaGUI {
    ... argument, resultat, b, f som før ...
    Choice menu = new Choice();

    String[] valuta = { "Euro", "US dollar", "Norske kroner",
                       "Svenske kroner" };
    double[] kurs = { 742, 697, 90, 86 };

    class Beregnlytter implements ActionListener {
        public void actionPerformed(ActionEvent e)
        { int fremmed = Integer.parseInt(argument.getText());
          int valutantnr = menu.getSelectedIndex();
          resultat.setText(Double.toString(fremmed * kurs[valutantnr] / 100)); }
    }

    public FlerValutaGUI() {
        b.addActionListener(new Beregnlytter());
        resultat.setEditable(false);
        for (int i=0; i<valuta.length; i=i+1)
            menu.addItem(valuta[i]);
        f.setLayout(new FlowLayout());
        f.add(menu); f.add(argument); f.add(b);
        f.add(new Label("Kroner: ")); f.add(resultat);
        f.pack(); f.show(); }
}
```