Genbrugelige komponenter og designmønstre i Java

Jacob Nordfalk

Ingeniørhøjskolen i København

Nykøbing F itvisioncenter

24. februar 2004

Program

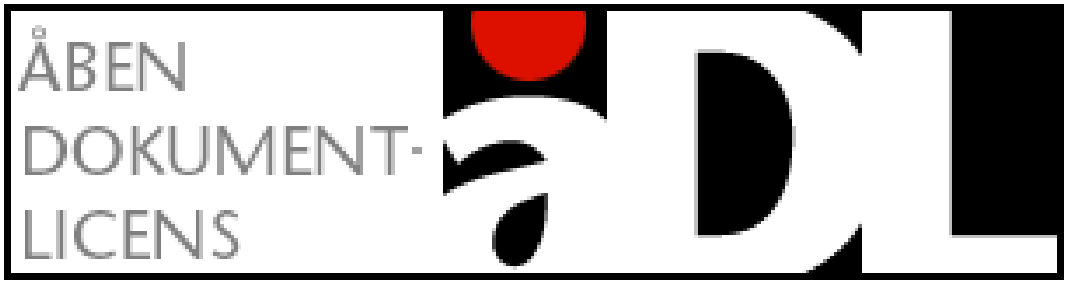
- Om Jacob Nordfalk
- introduktion (ikke-teknisk del)
  - Komponentbaseret programmering
  - Designmønstre i programmering
    - Eksempler (Prototype, Komposit, ...)
    - Evt. Model-View-Controller (MVC-arkitekturen)
- Pause
- Programmering (teknisk)
  - Designmønstre i et tegneprogram
  - Komponenter i Java (javabønner)

Om Jacob Nordfalk

- cand.scient. i fysik
- Udvikler for DSB
  - rejseplanen.dk og Byens Puls i Java
- Lektor på Ingeniørhøjskolen i København
- Forfatter
- Sprogligt interesseret med holdninger
  - Esperanto (internationalt sprog)
  - Træffende begreber på dansk øger forståelsen
- Tilhænger af åben kildekode (Open Source)
  - Inkarneret Linux-bruger
  - Bøger er (mest) under Åben Dokumentlicens

Åben Dokumentlicens (ÅDL)

- Dette foredragsmateriale er under ÅDL
  - Du har lov til frit at kopiere dette værk
  - Ændrer du i værket, skal (i hvert fald) de dele, der stammer fra dette værk, igen frigives under ÅDL
  - Licensens fulde ordlyd findes på http://www.sslug.dk/linuxbog/licens.html



Reklame

Indhold

- Komponentbaseret udvikling
- 20 designmønstre
- Javas standardbibliotek
- Java-teknologier

85% af bogen er under ÅDL og er gratis tilgængelig på http://javabog.dk/VP

Andre værker:
http://javabog.dk/JSP
http://javabog.dk/OOP

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

- Demonstration af brug at komponenter i et udviklingsværktøj

Mere om javabønner i den tekniske del efter pausen

Designmønstre

- Navngiven måde at programmere på
- Beskrivelse af, hvordan et bestemt slags problem kan løses
  - Beskrivelse af konsekvenserne af denne måde at løse problemet på.
- Det samme designmønster kan løse et lignende problem i et andet program
  - Engelsk: (Reusable) Design Pattern
    - Mønster = noget, som gentages
    - Design = hvordan et program er sat sammen
      - Begreb fra OOAD – Objektorienteret analyse og design
      - Designet vises ofte som et klassediagram

Formål med designmønstre

- At give en fælles begrebsramme
  - Lettere at forklare hvad man har lavet
  - Klar begrebsramme lettere at forstå
  - Lettere at beslutte hvordan noget laves
- Ikke "opfinde den dybe tallerken" igen.
  - Velafprøvede idéer til godt design
  - Undgå de mest almindelige faldgruber
- Mindske graden af bindinger (kobling) mellem forskellige dele af et program

Eksempler på designmønstre

- Singleton
  - sikring af, at der kun eksisterer ét objekt af en bestemt slags
- Prototype
  - objekter oprettes ud fra eksisterende skabelon-objekter
- Objektpulje
  - genbrug de samme objekter igen og igen ved at huske dem i en pulje
- Adapter
  - får et objekt til at passe ind i et system ved at fungere som omformer mellem objektet og systemet

Eksempler på designmønstre

- Iterator
  - hjælper med at gennemløbe nogle data
- Observatør/Lytter
  - 'abonnere' på, at en ting (hændelse) sker
- Komposit/Rekursiv Komposition
  - objekt, der kan indeholde andre objekter, inkl. sin egen slags
- Kommando
  - registrér brugerens handlinger, så at de kan fortrydes igen
- Model-View-Controller-arkitekturen
  - opdel programmet i en datamodel, en præsentation af data og en kontrol-del

Designmønstret Prototype

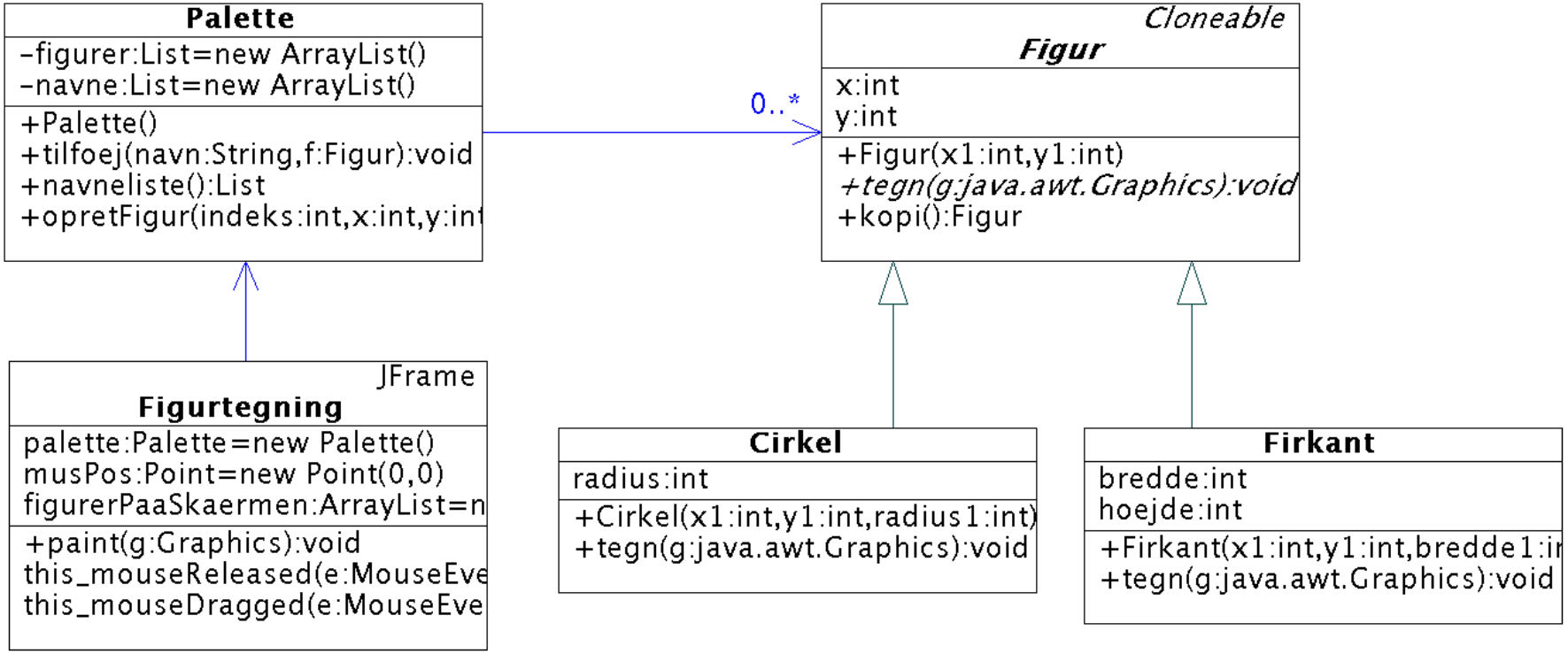
(objekter oprettes ud fra en skabelon)

Problem: Klienten ved ikke, hvad der skal oprettes, men kan dog angive et andet objekt, som ligner det, der skal oprettes

Løsning: Brug det andet objekt som Prototype, og opret objektet ud fra prototypen

Prototyper i et tegneprogram

- Palette har liste af figur-prototyper
  - Liste kan senere nemt udvides
- Bruger kan vælge i listen
  - Når der vælges i paletten, anvendes det pågældende element som Prototype til objektet, der skal tegnes på skærmen



Designmønstret Komposit

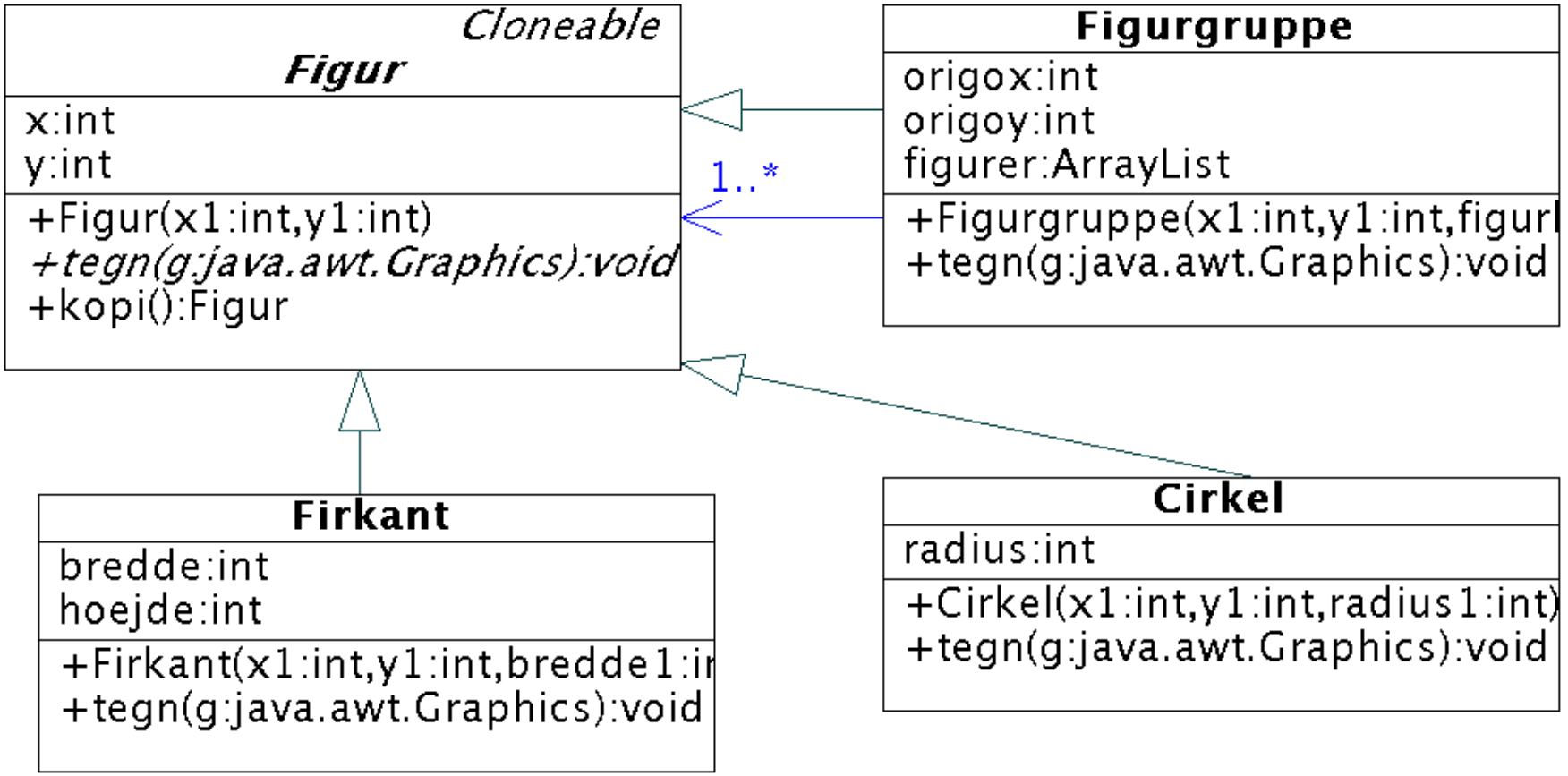
(eng.: Composite; objekt kan indeholde andre objekter, inkl. sin egen slags)

Problem: Hvordan kan man gruppere nogle objekter, så de kan behandles som ét objekt? Der er brug for grupperinger i flere niveauer

Løsning: Definér en fælles superklasse for alle enkeltobjekter, og definér en klasse til sammensatte objekter, som har en liste af de objekter den består af

Komposit i et tegneprogram

- Figurgruppe
  - Arver fra Figur
  - Har en liste af figurer



Designmønstret Kommando

(eng.: Command; registrér brugerens handlinger, så de kan fortrydes igen)

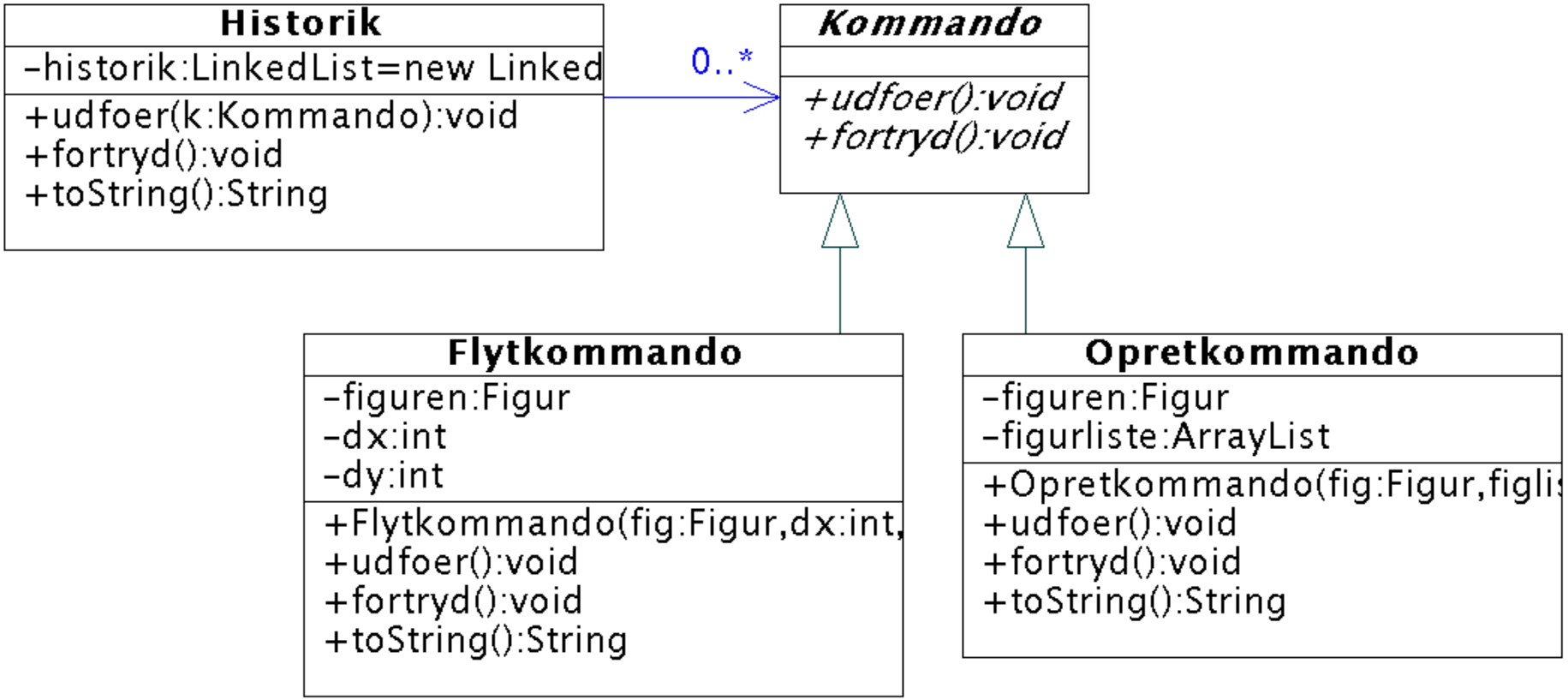
Problem: Brugeren skal kunne udføre og fortryde sine ændringer på data
(naiv løsning: Kopiér alle data ved hver ændring)

Løsning: Definér et Kommando-objekt, der repræsenterer ændringen

- ved, hvordan ændringen udføres
- husker de oprindelige værdier, sådan at ændringen kan fortrydes igen

Kommando i tegneprogram

- Kommando med udfør og fortryd
  - FlytKommando(husker ændring i x og y)
  - OpretKommando
- Kommandoer gemmes i historik



Model-View-Controller

Programmer med en brugergrænseflade kan inddeles i tre dele:

1.Modellen (data og de bagvedliggende beregninger)

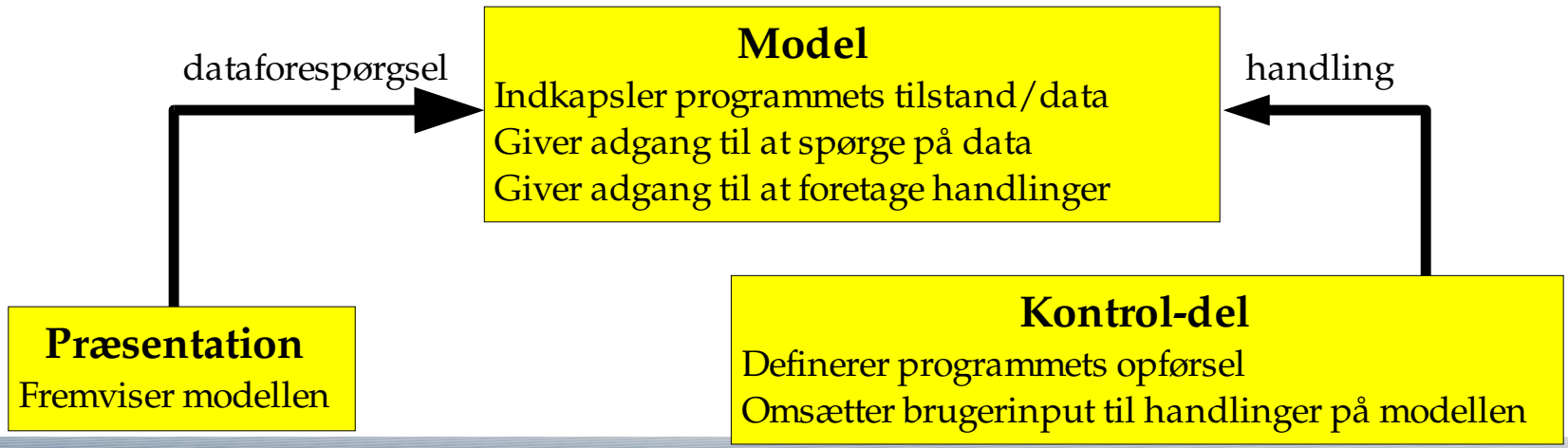
En bankkonto har navn på ejer, kontonummer, kort-ID, saldo, bevægelser, etc.
Saldo kan ikke ændres direkte, men med handlingerne overførsel, udbetaling og indbetaling.

2.Præsentationen af data over for brugeren,

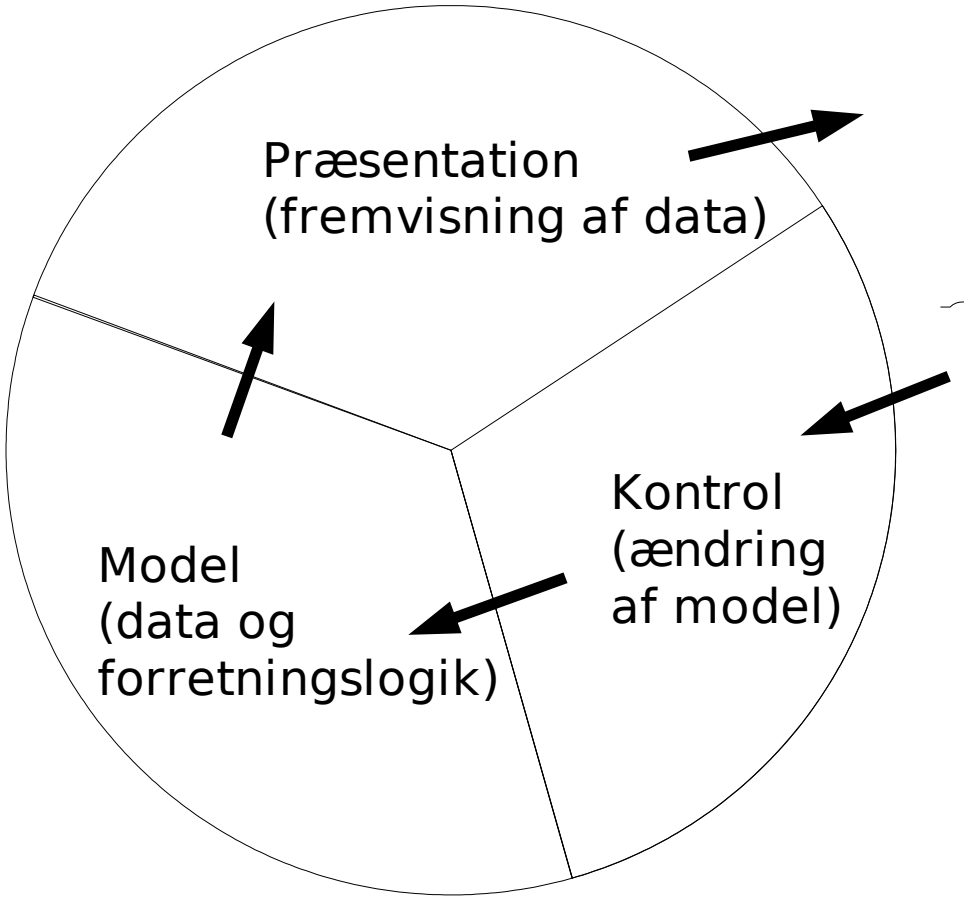
Bankkonti præsenteres meget forskelligt.
I en pengeautomat vises ingen personlige oplysninger overhovedet.
I et netbank-system kan saldo og bevægelser ses (det kunne være en webløsning i HTML).

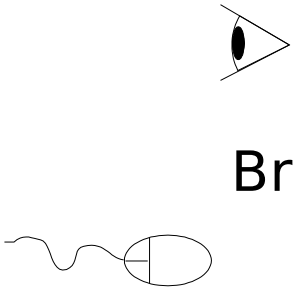
3.Mulighederne for at ændre i data gennem handlinger

I en pengeautomat kan man kun hæve penge.
I et netbank-system kan brugeren måske lave visse former for overførsel fra sin egen konto.
Ved skranken kan medarbejderen derudover foretage ind- og udbetalinger.



Model-View-Controller





Bruger

Genbrug af kode?

Flere præsentationer
af data samtidigt?

Muligheder for opdatering af præsentation:

A - Præsentationer undersøger modellen

B - Kontroldel underretter præsentationer

C - Modellen underretter præsentationer


Pause

Tegneprogrammets programkode

(Demonstration i JBuilder)

Komponentbaseret udvikling

Eksempel: Bønnen TextField

Egenskab	*Type*	*Sættes med*	*Læses 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);
  }
}
```

Komponentbaseret udvikling

En meget simpel bønne: GentagTekst

Herunder er en simpel grafisk bønne, der tegner en tekst tre gange skråt under hinanden.

```
package vp;
import java.awt.*;
public class GentagTekst extends Component
{
   private String tekstDerSkalVises = "gentag";

   public void setTekst(String t)
   {
      tekstDerSkalVises = t;
   }

   public String getTekst()
   {
      return tekstDerSkalVises;
   }

   public void paint(Graphics g)
   {
      g.drawString(tekstDerSkalVises, 0, 10);
      g.drawString(tekstDerSkalVises, 5, 15);
      g.drawString(tekstDerSkalVises, 10,20);
   }
}
```

Fra værktøjet:

```
import vp.*;
import java.awt.*;

public class VindueMedGentagTekst extends Frame
{
   GentagTekst gentagtekst1 = new GentagTekst();

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

   private void jbInit() throws Exception  {
      this.setSize(new Dimension(319, 247));
      this.setLayout(null);

      gentagtekst1.setTekst("ryst!");
      gentagtekst1.setBounds(new Rectangle(177, 190, 111,
      this.add(gentagtekst1, null);
   }
}
```

Ud fra kildeteksten ses, at bønnen har egenskaben *tekst* (der bestemmer, hvilken tekst der skal vises).

En javabønne siges at have en egenskab, hvis den har en tilsvarende get- og/eller set-metode

---

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

F.eks.:

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

Bønner bør være afgrænsede og uafhængige

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, og for hver egenskab en beskrivelse og hvordan den aflæses og 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

Bønner kan understøtte hændelses-lyttere

F.eks.:

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