Videregående programmering i Java

Dag 2 - Objektorienterede principper

Fastlæggelse af emne for projekt
Ansvarsområder, kobling og indkapsling
Programmere i pakker
Specificere funktionalitet i et interface
Datastrukturer: Afbildinger (hashtabeller og træer)

Læsning: VP 1.2-1.6, VP 15.3, VP 15.6-15.7

Dette materiale er under Åben Dokumentlicens, se http://www.sslug.dk/linuxbog/licens.html

Valgfrie emner

	Point	Emne
Lidt	35	Datastrukturers interne virkemåde og kørselstider
Jo, lidt!	11	Nye faciliter i Java: Generics, autoboxing
ja	42	Hvordan definere egne generics
ja	48	Mere om Swing og grafiske brugergrænseflader
ja	57	Kommunikation mellem processer og tråde i et GUI-program
nej	22	Styring af eksterne komponenter (OpenOffice.org / COM)
nej	23	Avanceret JDBC
Lidt	30	Objektpersistens og JDO - Java Data Objects
ja	41	Internationalisering (flersprogede programmer)
Lidt	29	JAR-filer og oprettelse af eksekverbare JAR-filer
nej	23	Optimeringsværktøjer (Borland OptimizeIt)
nej	20	J2ME, midletter og programmering af mobiltelefoner
Lidt	37	J2EE, EJB og serversystemer
nej	25	Værktøjer til forbedring af kodekvalitet - metrikker og audit
?	31	AOP - Aspekt-orienteret programmering
?		Dit eget forslag: SWT

Kursusopgaven

- Kursusopgaver fra sidste semester
  - Regnskabssystem m flere brugerflader
  - Regelbaseret produktkonfigurator
  - Generere .h-filer og dokumentation fra database
  - Netværksovervågningsværktøj: Nødstrømsanlæg, tavler
  - Forudsige farten af en sejlbåd (empirisk/modelforsøg)
  - Visuel GIS-konfigurationseditor (XML)
  - Racerbilspil over netværket
  - Grafisk overvågning af driftsdata
  - Visuel formel-editor (a la Word)

Kursusopgaven

- Idéer i dette semester
  - Kig uddelte udskrift igennem

  Det er OK at bygge videre på et gammelt projekt

Kursusopgave

- Runde i plenum
  - Hvad kunne du tænke dig at lave projektopgave om?
  - Er du åben over for flere deltagere?
- Diskussion to og to
  - Udfærdige/diskutere krav til hvert af projekterne
  - Diskutere mulige teknologier
  - Diskutere om eksisterende (Open Source-)projekt kan danne udgangspunkt eller komponent herfra kan hjælpe
- Hjemmeopgave fra sidste gang:
  - Beskriv din ide til kursusopgave (10-30 linier)

Vector-klassens interne virkemåde

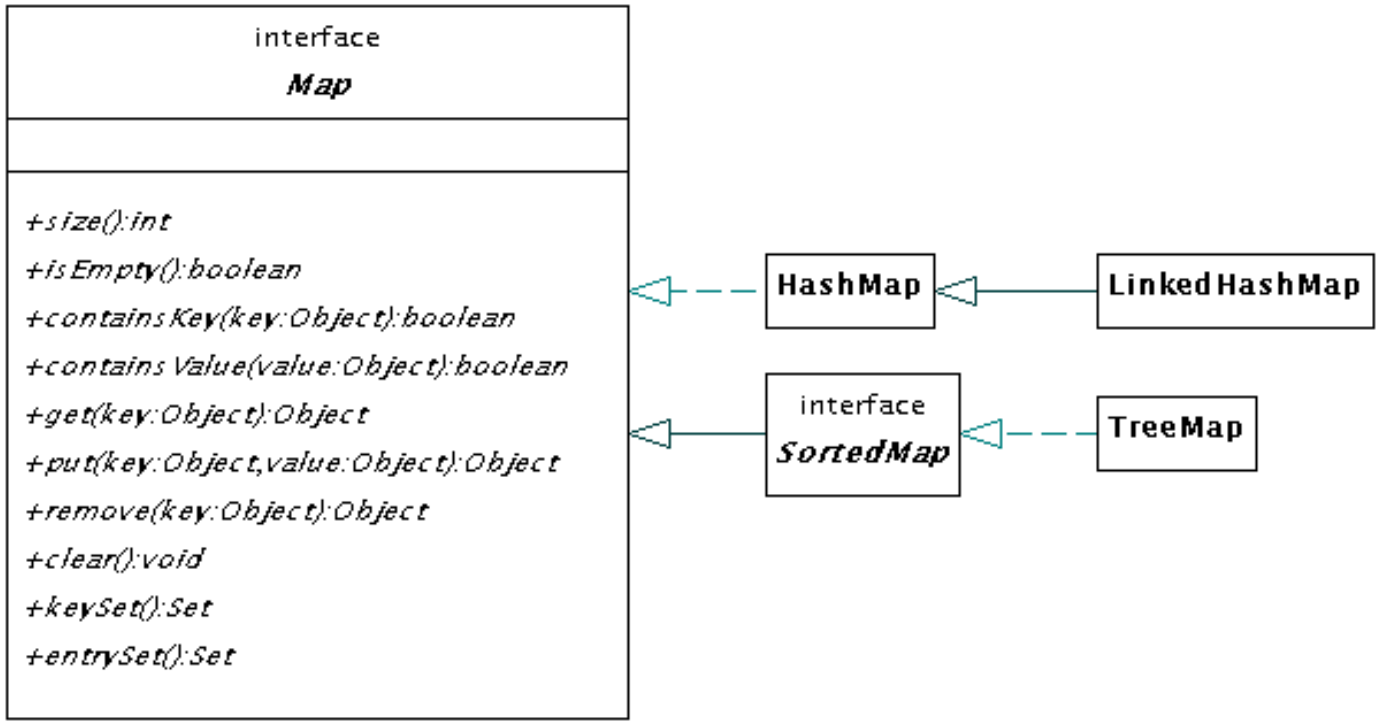
- Hvordan fungerer Vector-klassen internt?
- variabler
- metoder
  - isEmpty()
  - firstElement()
  - elementAt()
  - addElement()
  - removeElementAt()

Afbildninger (nøgleindekserede lister)

- En Map (da.: afbilding) beskriver nøgle-værdi-par

```
HashMap ord = new HashMap();
ord.put("estas", "(det) er");
ord.put("bona", "god");
ord.put("granda", "stor");
ord.put("longa", "lang");
ord.put("hundo", "hund");

esperantoOrd = "granda";
// slå esperantoordet op og få det danske ord
danskOrd = (String) ord.get( esperantoOrd );
```

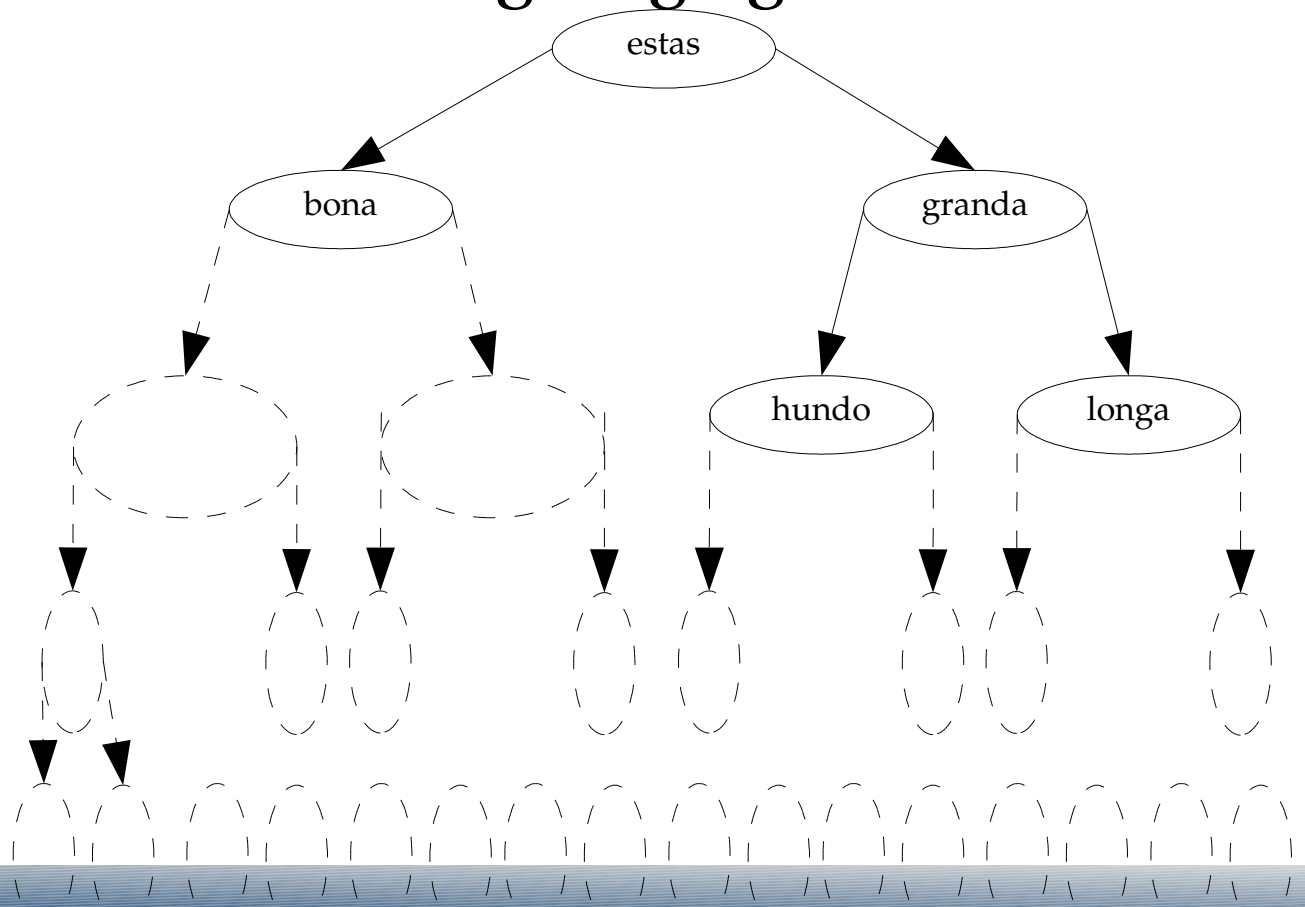


Hashtabeller

- hashCode() beregner *hashkoden* af et objekt
  - en 'næsten' unik nøgle, beregnet ud fra objektets data
    - et andet objekt med samme data har samme hashkode
    - andre objekter har en anden hashkode (i 99% af tilfældene)
  - alle de almindelige klasser har en god hashCode()-metode
- hashkoden/nøgleværdien anvendes som indeks i et (stort) array, hvor elementet gemmes.
  - Hvis man vil slå et bestemt objekt (nøgle) op, beregnes dens hashværdi og der slås op på det pågældende indeks

Træer

- I et binært søgetræ har hver indgang en reference til
  - en indgang med lavere værdi og
  - en indgang med højere værdi
- Indgangene er altid sorteret efter værdi
- $n = 2^{\text{antal lag}} - 1 \quad \Rightarrow \quad$ søgning og indsættelse er O(log(n))



Kørselstider og Store-O-notationen

Store-O

tiden en operation tager som funktion af antal elementer n

- O(1) - konstant i tid
  - Eksempel: Slå et element op i et array
- O(log(n)) - logaritmisk
- O(n) - proportionalt
  - Eksempel: Gennemløbe alle element i et array
- $O(n^2)$ - kvadratisk (hmm...)
- $O(e^n)$ - eksponentielt (uha!)

Kørselstider og Store-O-notationen

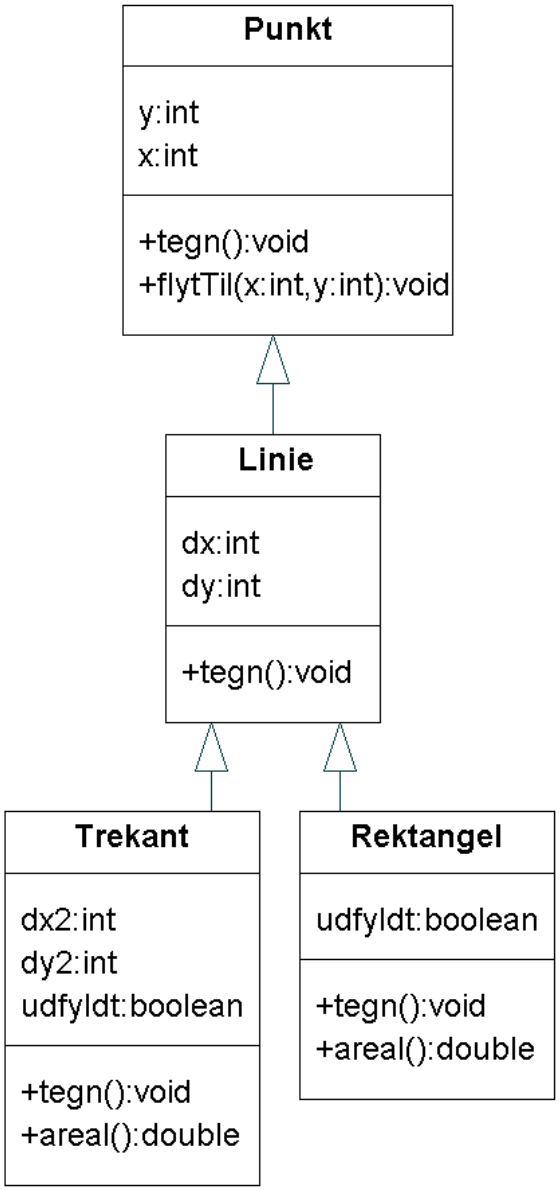
Opgave: Find kørselstid for LinkedList, ArrayList, TreeSet, HashSet, for indsættelse, sletning, opslag, søgning

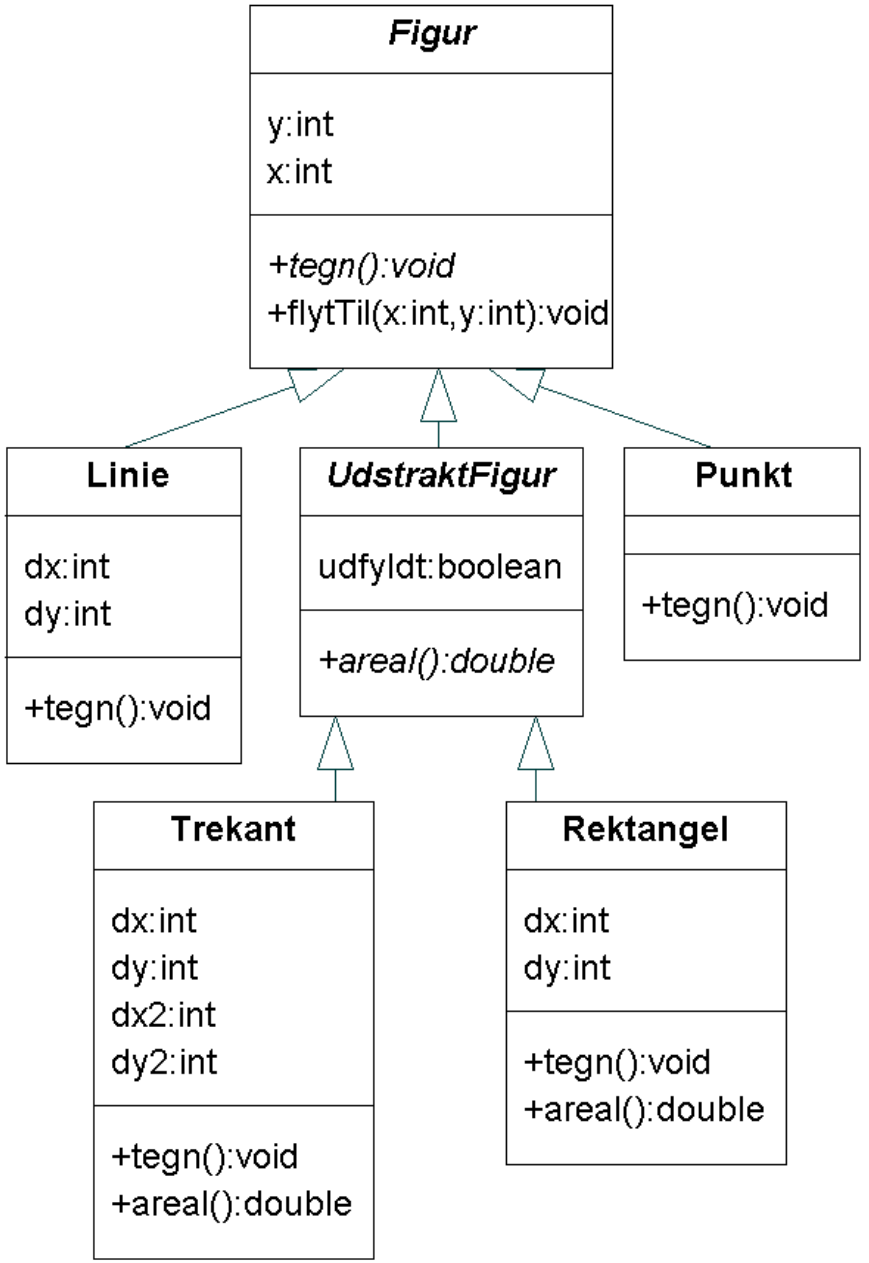
- O(1) - konstant i tid
  - Eksempel: Slå et element op i et array
- O(log(n)) - logaritmisk
- O(n) - proportionalt
  - Eksempel: Gennemløbe alle element i et array
- $O(n^2)$ - kvadratisk (hmm...)
- $O(e^n)$ - eksponentielt (uha!)

Evt.: Holdninger til arv (VP 15.1)

- To holdninger til arv
  - Den "kodenære"
    - Arv bruges til at genbruge variabler og metoder. Hvis to klasser har fælles variabler eller metoder, bør man lave en fælles superklasse, der tager vare på de ting, der er fælles.
    - Formål: spare kode
    - 'nedefra-og-op'-strategi
  - Den "analytiske"
    - Arv repræsenterer en er-en-relation, og nedarving bør ske mellem klasser, når de begreber, som de står for, har en 'er-en'-relation til hinanden.
    - Formål: analyse, overskuelighed, klarhed
    - 'oppefra-og-ned'-strategi

Evt.: Holdninger til arv (VP 15.1)





Evt.: Delegering i stedet for arv (VP 15.2)

- (droppes - behandles måske senere i kurset)

Modularitet

- Programmer er ofte for store og for komplekse til at blive designet og implementeret som ét system.
- De bør derfor deles op i håndtérbare moduler
  - "Del og hersk"
- Principper for modulerne:
  - Hvert modul bør have et klart ansvarsområde
  - Modulerne bør have lav kobling indbyrdes

Ansvarsområder

Ansvarsområder for et objekt/modul kan være:

- At oprette nye objekter eller udføre en beregning
- At foretage en handling i andre objekter
- At kontrollere og koordinere aktiviteter i andre objekter
- At kende private data
- At kende relaterede objekter
- At kende ting, som objektet kan beregne

- Høj kohæsion (sammenhæng - eng.: high cohesion)
  - at et objekt har ét overskueligt og let forståeligt ansvarsområde (eller eventuelt flere områder tæt relaterede til hinanden)

Kohæsion og kobling

- Kohæsion
  - Hvor relaterede funktionerne, der varetages inden for et modul, er
  - Metoderne i en klasse bør gøre relaterede ting
  - Ergo: Høj kohæsion er godt
- Kobling
  - Handler om graden af afhængighed, der er mellem modulerne
  - Modulerne bør stort set være uafhængige og kun løst forbundet
  - Ergo: Lav kobling er godt

Indkapsling

- Veldesignede moduler indkapsler (eng.: encapsulate) informationerne og skjuler deres repræsentation for andre moduler ("klienter"), der bruger modulet
- "Need to know"-princip
  - Hvis et modul ikke har behov for at vide det, får det det ikke at vide
- Klienter skal vide **hvad** et modul gør, ikke **hvordan** det gør det

- Princip:
  - Adskil et moduls (et objekts) implementation fra grænsefladen (interfacet) til modulet.

Indkapsling med pakker

Adgang til variabler og metoder i et objekt:

Adgang	**public**	**protected**	**(ingenting)**	**private**
i samme klasse	ja	ja	ja	ja
klasse i samme pakke	ja	ja	ja	nej
nedarving i en anden pakke	ja	ja	nej	nej
ej nedarving og i en anden pakke	ja	nej	nej	nej

Holder man sig inden for samme pakke, er der altså *ingen* forskel mellem public, protected og ingenting.

- Et modul, der består af flere klasser, kan lægges i sin egen pakke
  - metoder/variabler, der er interne for modulet erklæres med pakke-synlighed (ingenting)
  - modulets klasser indkapsles derved, så at klasserne kan tilgå hinandens metoder/variabler, mens de ikke er synlige udefra

Interfaces

En klasse kan begrebsmæssigt opdeles i:

1) *Grænsefladen* - hvordan objekterne kan bruges udefra.
   Dette udgøres af navnene på metoderne, der kan ses udefra.

2) *Implementationen* - hvordan objekterne virker indeni.
   Dette udgøres af variabler og programkoden i metodekroppene.

Et 'interface' svarer til punkt 1): En definition af, hvordan objekter bruges udefra.
Man kan sige, at et interface er en "halv" klasse.

Et interface er en samling navne på metoder (uden krop)

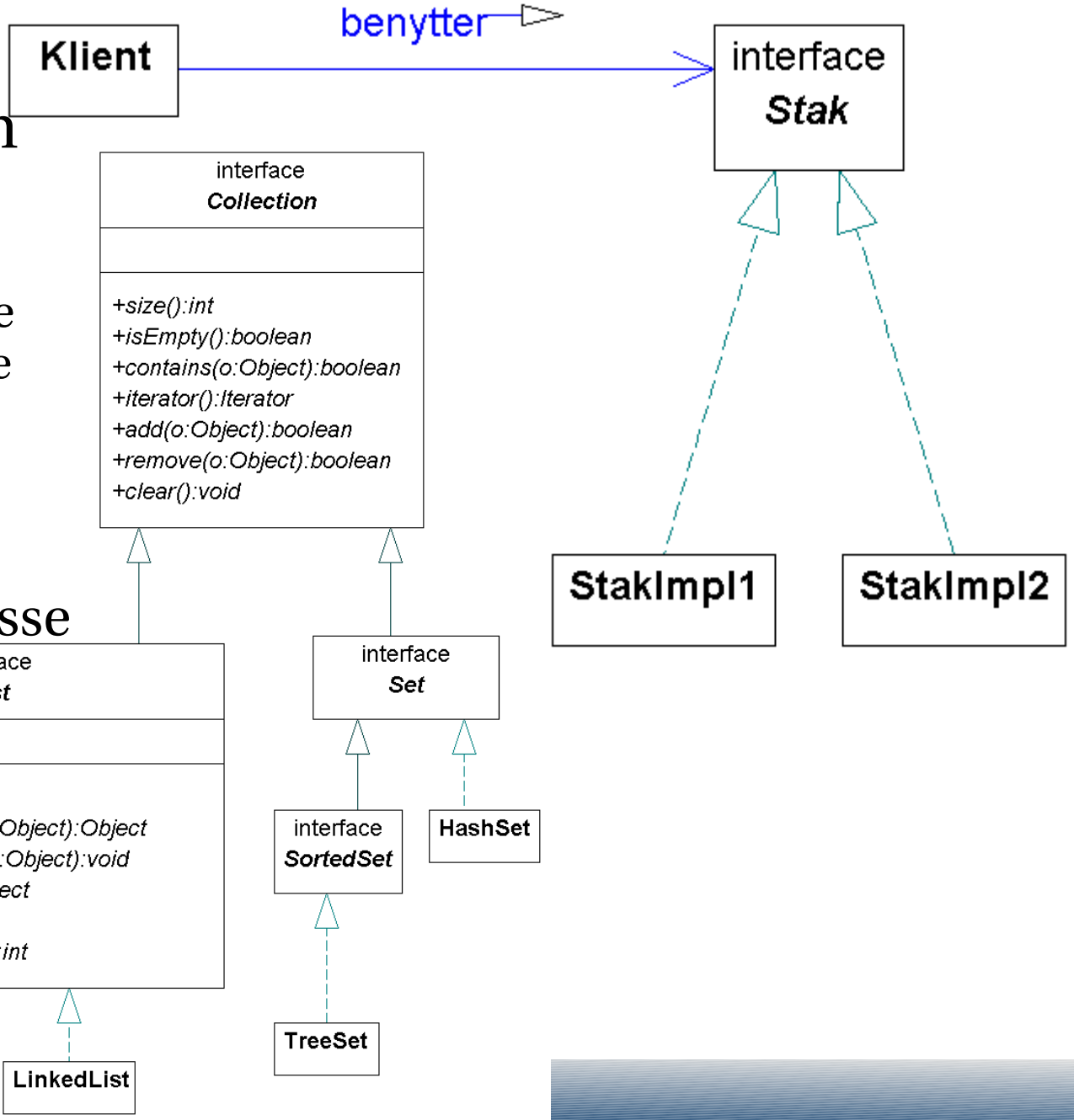
```
public interface ActionListener
{
    public void actionPerformed(ActionEvent e);
}
```

```
public interface Tegnbar
{
        public void sætPosition(int x, int y);
        public void tegn(Graphics g);
}
```

```
public interface MouseListener
{
    public void mousePressed(MouseEvent e);
    public void mouseReleased(MouseEvent e);
    public void mouseClicked(MouseEvent e);
    public void mouseEntered(MouseEvent e);
    public void mouseExited(MouseEvent e);
}
```

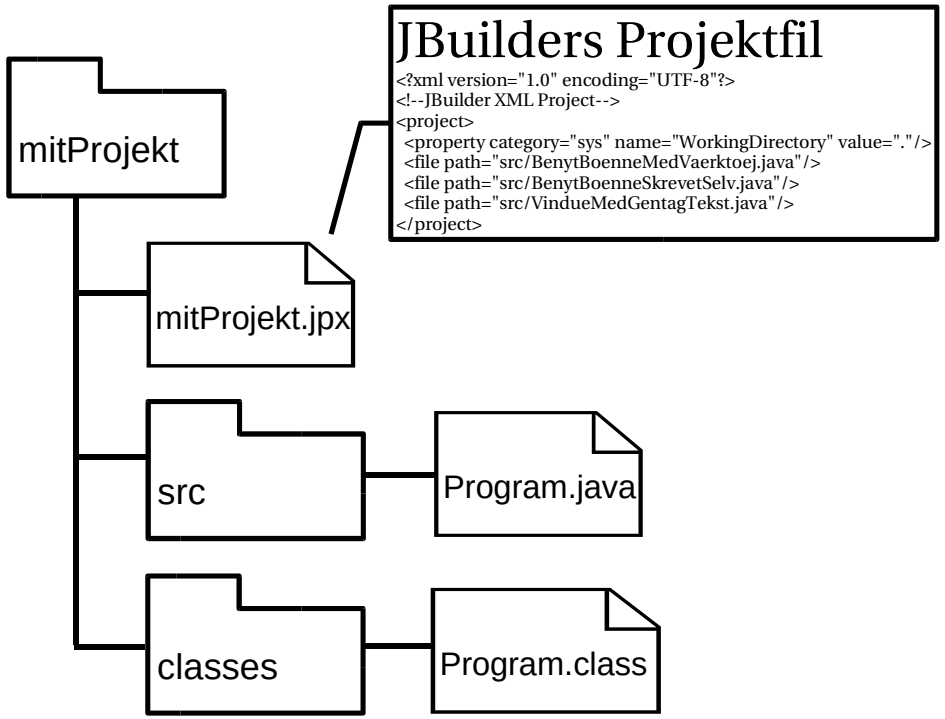
Specificere funktionalitet i interface (VP 15.3)

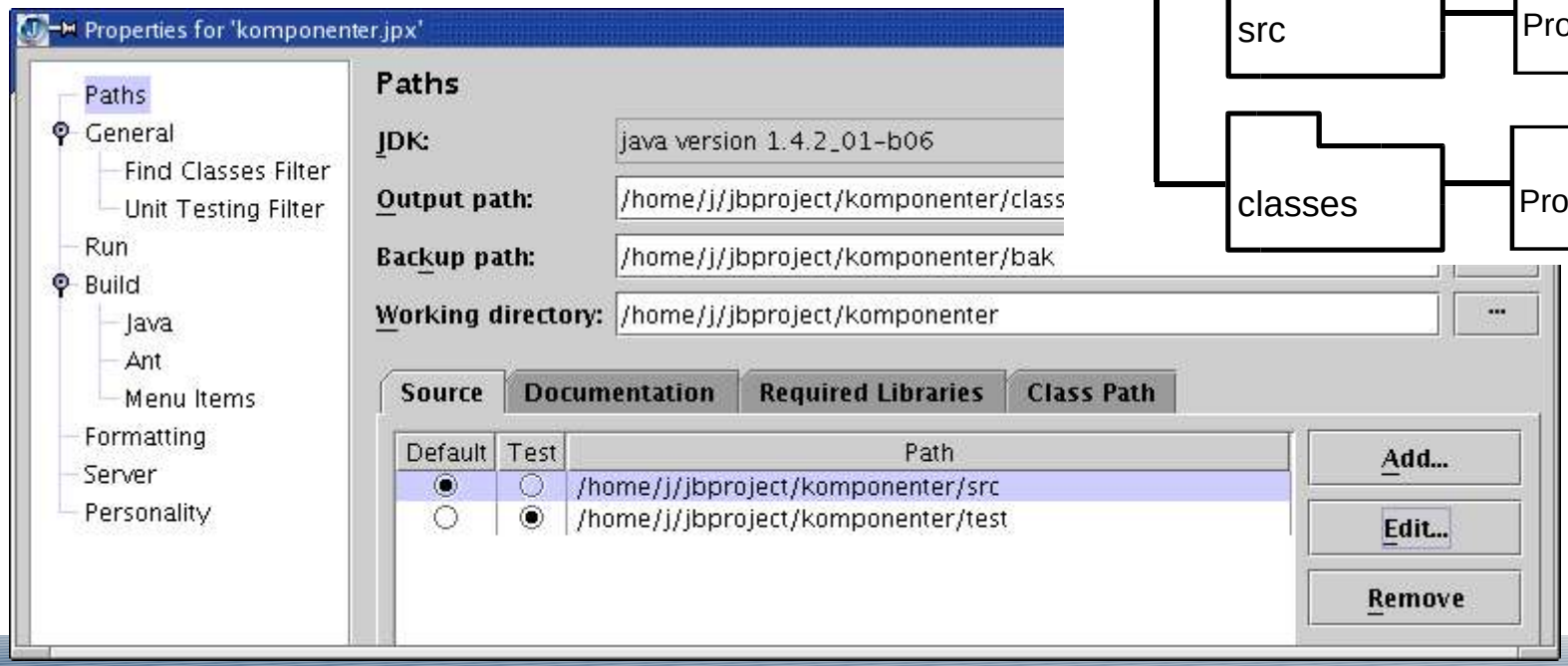
- Adskil grænsefladen fra implementeringen
  - Klarhed
  - Lav kombling
    - F.eks. mulighed for andre implementeringer senere
- Hvordan
  - I Java: interface
  - I C++: rent abstrakt klasse
    - multipel arv



JBuilder og projekter

- Tjek at projektet forventer kildetekst det rigtige sted!
  - Tjek stierne i projektet
    - Åbn "Project Properties"
    - Kildekode i src/
    - .class-filer i classes/
  - Sti på .java-fil = src/ + pakke





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
  }

  ...
}
```