Videregående programmering i Java

VPJ - dag 1

Introduktion og overblik over kurset
Datastrukturer: Lister, mængder og iteratorer
Opsamling: Grafikprogrammering, komponenter, containere og layout
Bruge komponenter (Javabønner)

Om kurset

- I kurset benyttes JBuilder
  - Brug fra CD eller hent fra http://borland.com/
  - Andre værktøjer kan også benyttes
- Lærebogsmateriale
  - "Videregående programmering i Java" af Jacob Nordfalk
    - En stor del af lærebogen (+opdateringer) findes på nettet: http://javabog.dk/VP/
  - Foredrag fra tidligere semestre om emnerne: http://javabog.dk/VP/lyd/
  - Præsentationerne kan efterfølgende findes på: http://javabog.dk/foredrag/VPJ-1_E2005/
  - Uddelte fotokopier

Kursets indhold

- Designmønstre
  - Singleton, Proxy, Fabrik, Objektpulje, Iterator, Adapter, Kommando, Model-View-Controller-arkitekturen (MVC),...
  - Flertrådet programmering og designmønstre herom
- Grafiske brugergrænseflader
- Komponentbaseret udvikling
  - definere egne grafiske komponenter (javabønner)
- Datastrukturer som lister, mængder og afbildninger
- Videregående emner inden for programmering
  - JNI: kald til maskinkode/C/C++ fra Java
  - Introspektion af klasser under kørsel
  - Java2D - avanceret grafiktegning
  - Dokumentation med javadoc-værktøjet
- Andre emner kan tages op efter deltagernes ønske.

Kursusopgave

- Kun én vej til at blive erfaren: At lave et større program!
  - Omfang: 80 timer eller mere (!)
  - Formål:
    - at deltagerne får programmeret så meget som muligt på et projekt
    - at de får arbejdet med de ting fra kurset der interesserer dem mest
    - en mindre del (ca. 30 %) af de berørte teknikker og designmønstre indgår
  - Kan laves i grupper á 1-3 personer
- Du skal lægge dig fast på et projekt **til gang 2 eller 3**
  - Undervisning i designmønstre er baseret på at du er i gang!
- Det kunne f.eks. være:
  - et projekt fra arbejdet
  - en prototype på dit afgangsprojekt
  - en visuel formel-editor (a la Word)
  - ...
- Hjemmeopgave
  - Beskriv din ide til kursusopgave (10-30 linier).

Valgfrie emner

Giv mellem 0 og 10 point til hvert emne:

- Datastrukturers interne virkemåde og kørselstider
  - Nye faciliter i Java: Generics, autoboxing
- Hvordan definere egne generics
- Mere om Swing og grafiske brugergrænseflader
- Kommunikation mellem processer og tråde i et GUI-program
- Styring af eksterne komponenter (OpenOffice.org / COM)
- Avanceret JDBC
  - Objektpersistens og JDO - Java Data Objects
- Internationalisering (flersprogede programmer)
- JAR-filer og oprettelse af eksekverbare JAR-filer
- Optimeringsværktøjer (Borland OptimizeIt)
- J2ME, midletter og programmering af mobiltelefoner
- J2EE, EJB og serversystemer
- Værktøjer til forbedring af kodekvalitet - metrikker og audit
- AOP - Aspekt-orienteret programmering
- Dit eget forslag:_____

UML og klasserelationer

Har-relationen

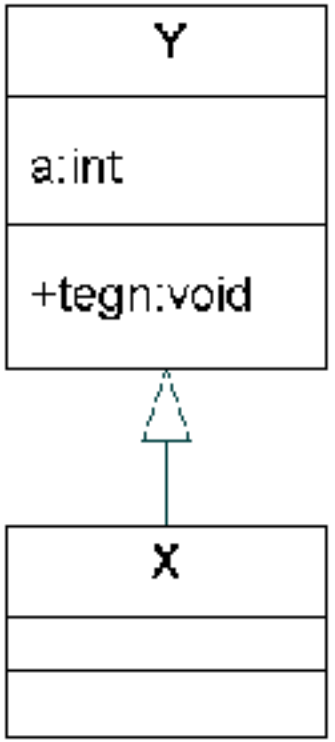
At X *har* Y betyder: i klassen X er defineret en variabel af type Y. Et X-objekt har derfor en reference til et Y-objekt, og derfor har X-objektet mulighed for at kalde metoder og bruge variabler fra Y-objektet gennem denne reference. Y-objekter kender ikke nødvendigvis noget til X-objekters eksistens.

Eksempel (det væsentlige i fed):

```
public class X
{
   private Y yobj = new Y();
   ...
}
```

Er-en-relationen

At X *er-en* Y betyder: i klassen X er defineret, at X arver fra klassen Y. Et X-objekt indeholder derfor (mindst) alle de metoder og variabler, som et Y-objekt indeholder.

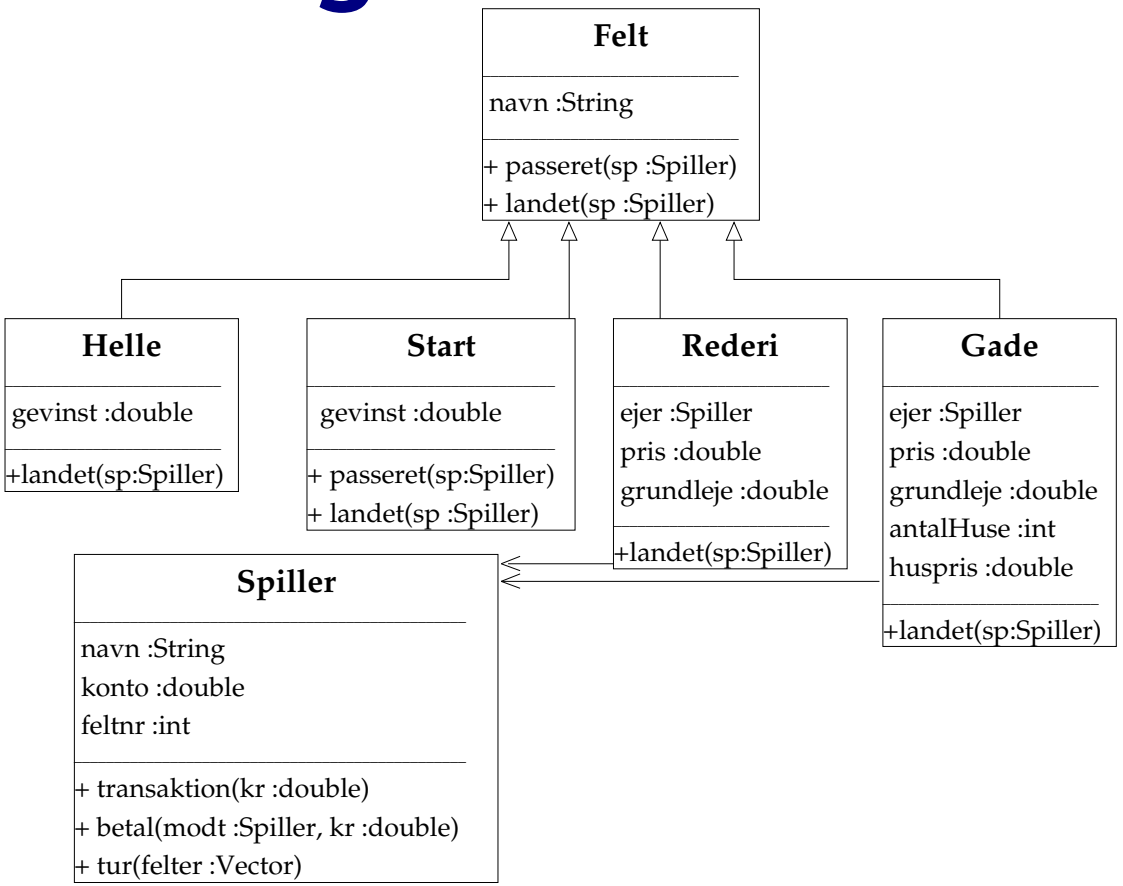


Eksempel:

```
public class X extends Y
{
   ...
}
```

Det er også en er-en-relation, når man implementerer et interface.

UML og klasserelationer



Rederi og Gade *har en* Spiller.

Helle, Start, Rederi og Gade *er-et* Felt.

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

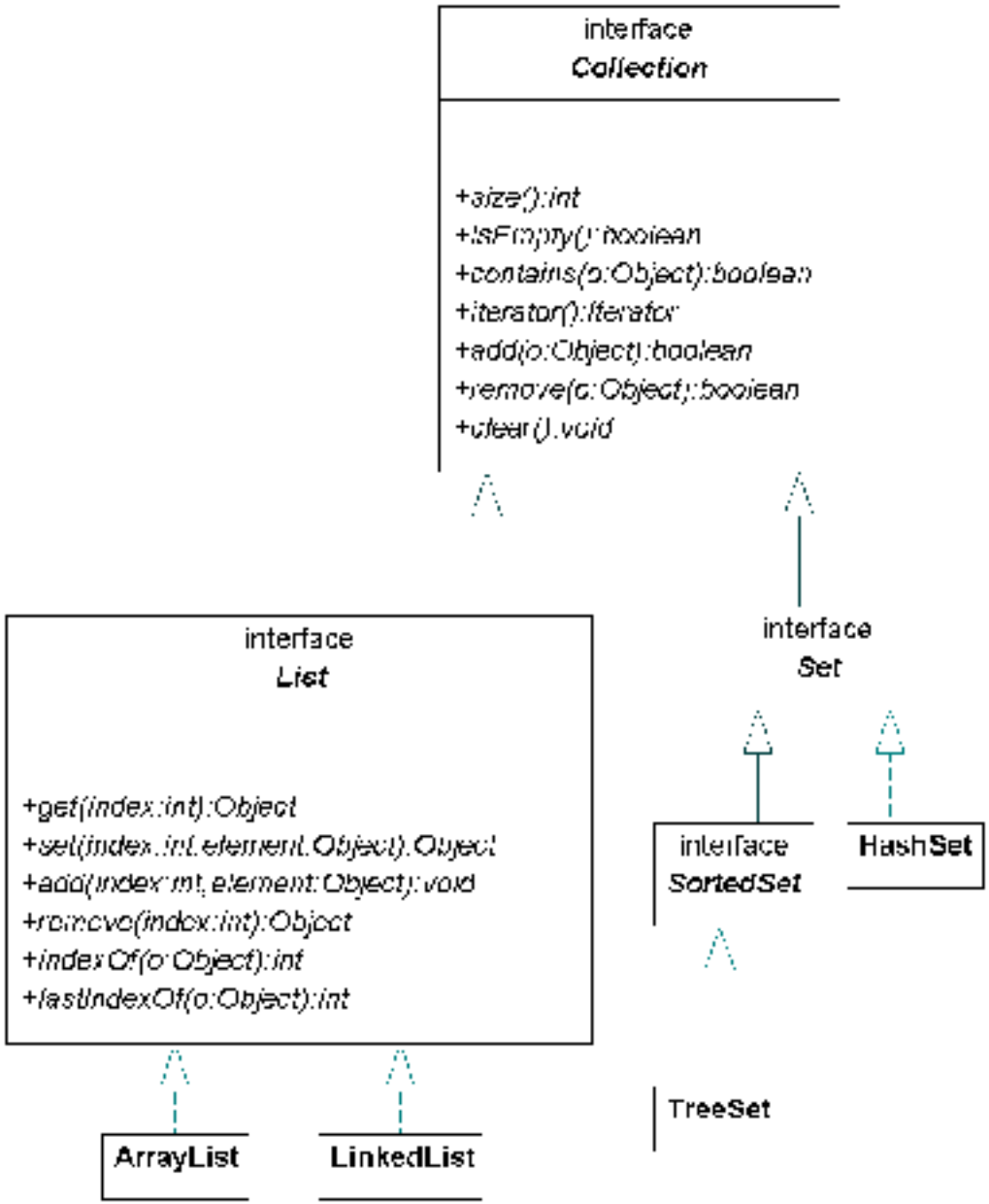
- At give en fælles begrebsramme
  - Lettere at forklare hvad man har lavet
  - Klar begrebsramme lettere at forstå
  - Lettere at beslutte hvordan noget laves
- Ikke "opfinde den dybe tallerken" igen.
  - Velafprøvede idéer til godt design
  - Undgå de mest almindelige faldgruber
- Mindske graden af 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
- Kommando
  - registrér brugerens handlinger, så at de kan fortrydes igen
- Observatør/Lytter
  - 'abonnere' på, at en ting (hændelse) sker

Samlinger af data

- Collection: Samling af data
  - Indeholder metoder fælles for alle datastrukturer
- List: Ordnet liste
  - To klasser implementerer List: ArrayList og LinkedList.
- Set: Uordnet mængde
  - HashSet implementerer Set.
- SortedSet: Sorteret mængde
  - TreeSet implementerer Set.



Samlinger af data

Eksempel på brug af liste

```
List liste;
liste = new ArrayList();
liste.add("æh");      // alle samlinger af data
liste.add("øh",0);   // kun lister
```

Gennemløb v.hj.a. tællevariabel

```
for (int i=0; i<liste.size(); i++) {
          String s = (String) liste.get(i);
          // gør noget med s
          System.out.println(s);
}
```

Et (lille) objekt, der hjælper med at gennemløbe data

Gennemløb v.hj.a. *iterator*

```
for (Iterator iter=liste.iterator(); iter.hasNext(); ) {
          String s = (String) iter.next();
          // gør noget med s
          System.out.println(s);
}
```

Man kan få et iterator-objekt ved at kalde iterator()
på enhver datastruktur (Collection)

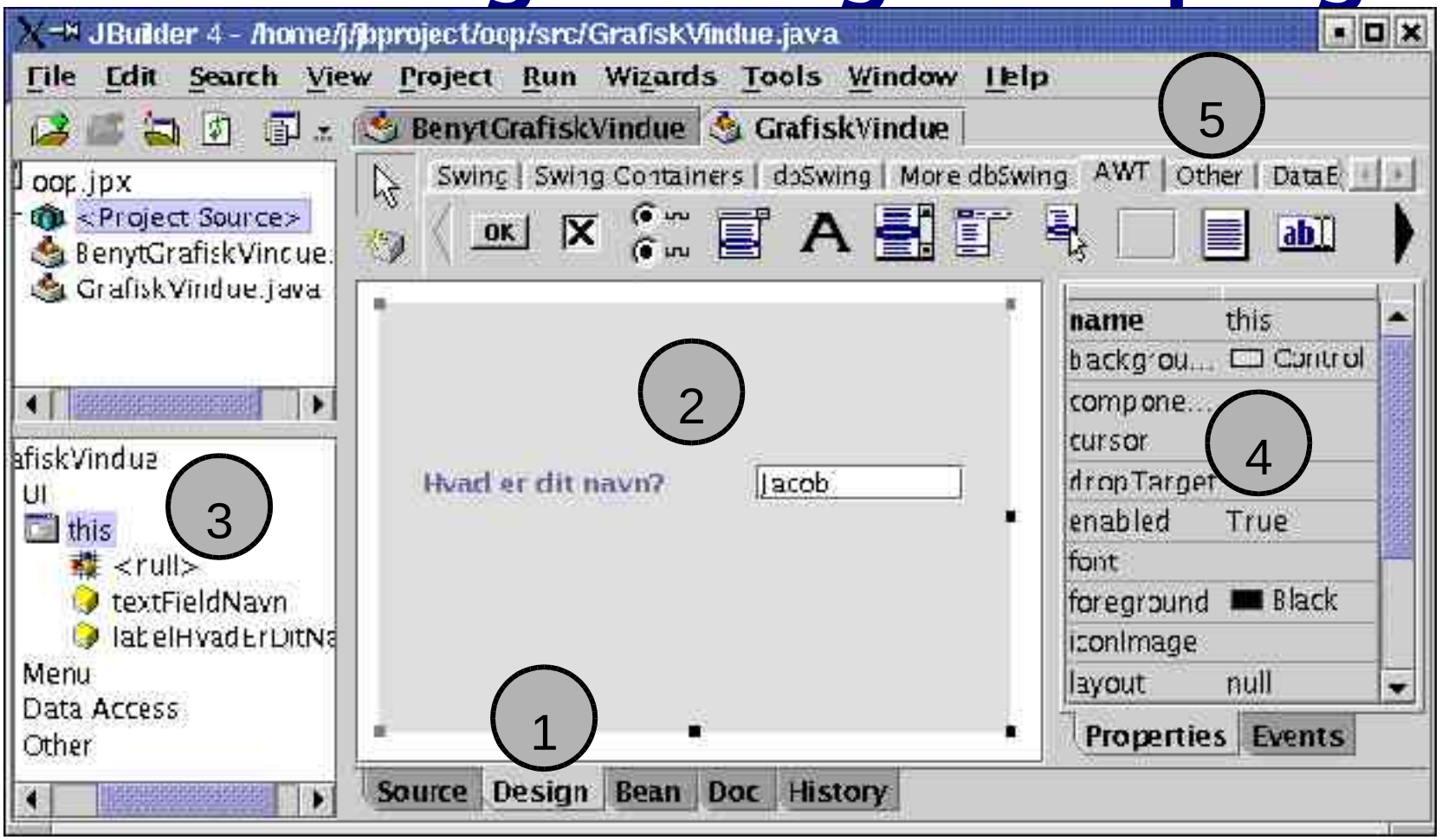
Kørselstider og Store-O-notationen

Store-O

tiden en operation tager som funktion af antal elementer n

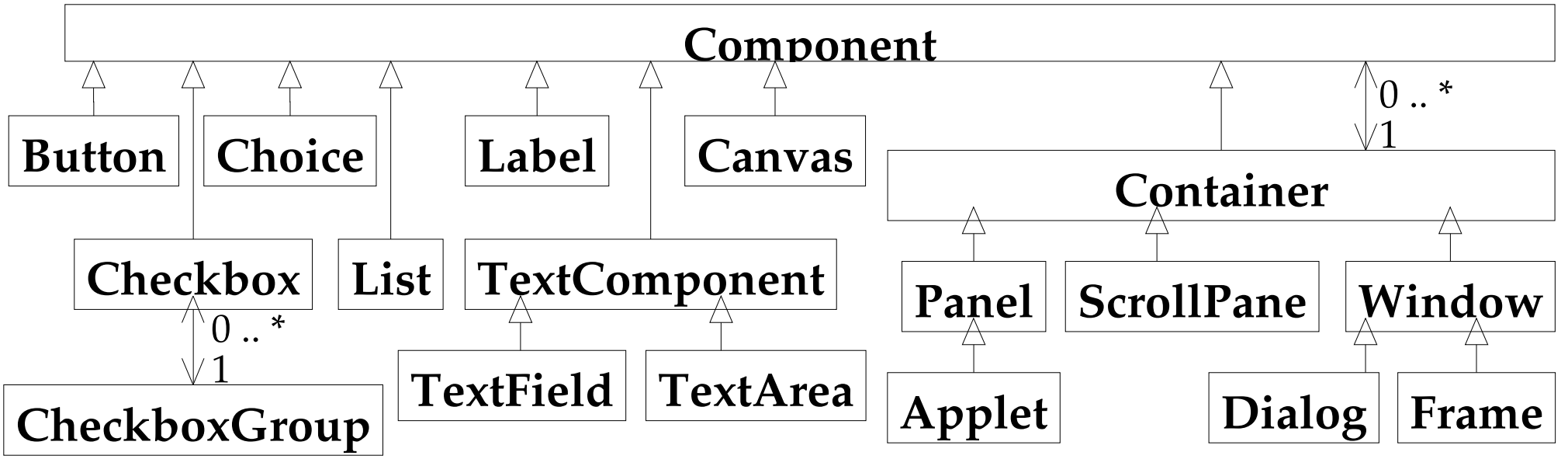
- O(1) - konstant i tid
  - Eksempel: Slå et element op i et array
- O(log(n)) - logaritmisk
- O(n) - proportionalt
  - Eksempel: Gennemløbe alle element i et array
- $O(n^2)$ - kvadratisk (hmm...)
- $O(e^n)$ - eksponentielt (uha!)

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

Komponenter og containere (AWT)



- Hule pile: *er-en*-relationer (dvs. nedarvning)
- De andre pile: *har*-relatioer
  - En container har nul til mange komponenter
  - Hver komponent har/tilhører én container
  - Tilsvarende med CheckboxGroup og Checkbox

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

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

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.

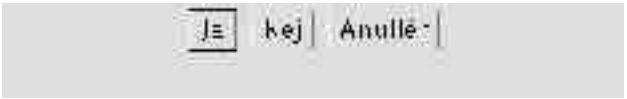
Løsning: Lad lytterne (observatørerne) implementere et fælles interface (eller arve fra en fælles superklasse) og registrere sig hos det observable (observerbare) objekt. Det observable objekt kan herefter underrette lytterne gennem interfacet, når der er brug for det.

Designmønstret Observatør/Lytter

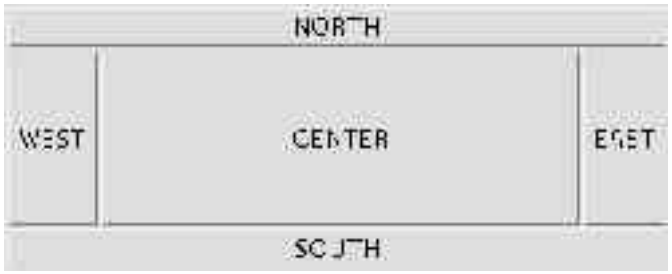
- Programkode til observabelt objekt (og javabønne)

Layout-managere

FlowLayout



BorderLayout



GridBagLayout

Komponentbaseret udvikling

- Demonstration af brug at komponenter i et udviklingsværktøj