

Læringsmål for forelesningen

- Objektorientering
 - Sammensatte objekter: Komposisjon og aggregering
 - Delegeringsteknikken
 - Delegering vs. Arv



Dagens forelesning

- Introduksjon og motivasjon
 - Hvorfor forelese om standardteknikker, såkalte patterns?
 - Hva slags problem løses med delegering?
 - Delegering vs. arv
- Delegeringseksempler
 - Person.children-liste
 - Sjefen og den som gjør jobben
 - Kokk og medhjelpere
 - Hierarkiske innstillinger (denne er hard)
 - ...

Bortsett fra arv...
så kan dere det viktigste om Java-språket!

Men... programmering er mer enn Java, dere
trenger også lære om standard kodingsteknikker,
f.eks. god strukturering av koden og bruk av
flere API'er

Standard kodingsteknikker

- “a design pattern is a general repeatable solution to a commonly occurring problem in software design”
- Gjenbrukbare kodingsteknikker som har vist seg å være praktiske/effektive løsninger
- Standardteknikker er ofte om såkalte samhandlingsmønstre – fordeling av logikk på flere objekter
 - **delegering** – et objekt får hjelp fra andre objekter
 - **observatør** – **observert** – et objekt oppdateres om hva et annet objekt gjør

Komposisjon og aggregering

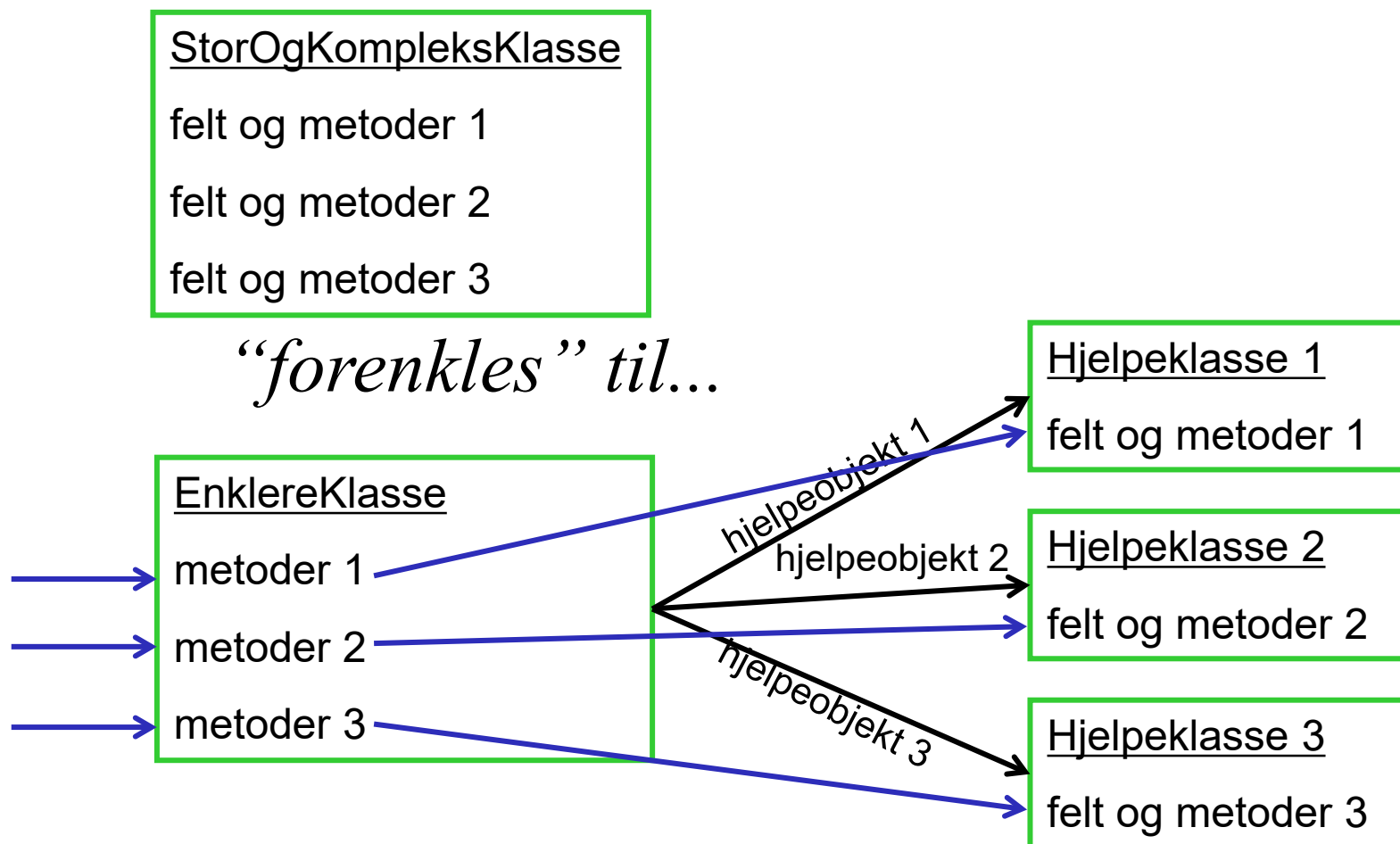
- Aggregering: *har-en*-relasjon
 - A *har en* B
 - A og B kan leve uavhengig av hverandre
 - Svak kobling
 - Kardinalitet?
- Komposisjon: *del-av*-relasjon (*hører-til*)
 - Spesialtilfelle av aggregering
 - B er *en del av* A, B uten A gir lite mening (eks: rom i hus)
 - Når A “dør”, så “dør” samtidig B (hva med motsatt?)
 - Sterkere kobling enn aggregering
 - Kardinalitet?
- I tillegg har vi Arv – *er-en*-relasjon, som vi ser på til slutt.

Delegering

- Et objekt fremstår med en ferdighet, men overlater til et annet (internt) objekt å gjøre jobben
- Relasjonen kan være via komposisjon eller aggregering
- (Matlagings)analogi
 - (kjøkken)sjefen har ansvaret, og har (kun) en koordinerende rolle overfor underordnede (kjøkken)medarbeidere
 - når kjøkkensjefen tar på seg et oppdrag (en oppskrift), så kan hun fremstå som å kunne utføre det, fordi (en av) medarbeiderne kan gjøre det for henne, dvs. få delegert oppgaven til seg
- Mer fleksibel teknikk enn arv for gjenbruk av eksisterende funksjonalitet

Liten digresjon...

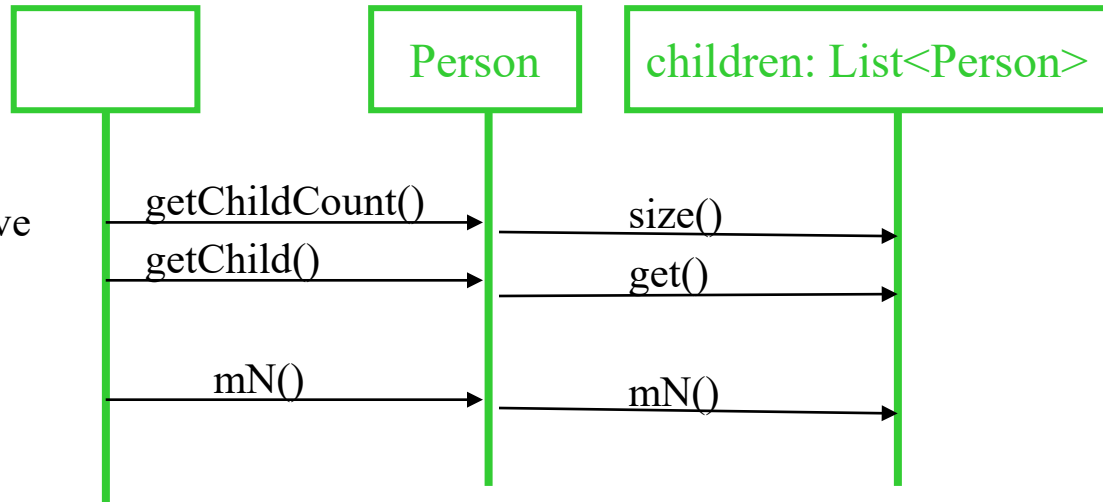
- Delegering ligner litt på en teknikk en ofte bruker for å forenkle store klasser:



Eksempel: Person.children-liste

- Person-objekt delegerer til List<Person>-objekt
 - Delegerende: Person
 - Delegat: List<Person>
- Metoder m1.. mN:

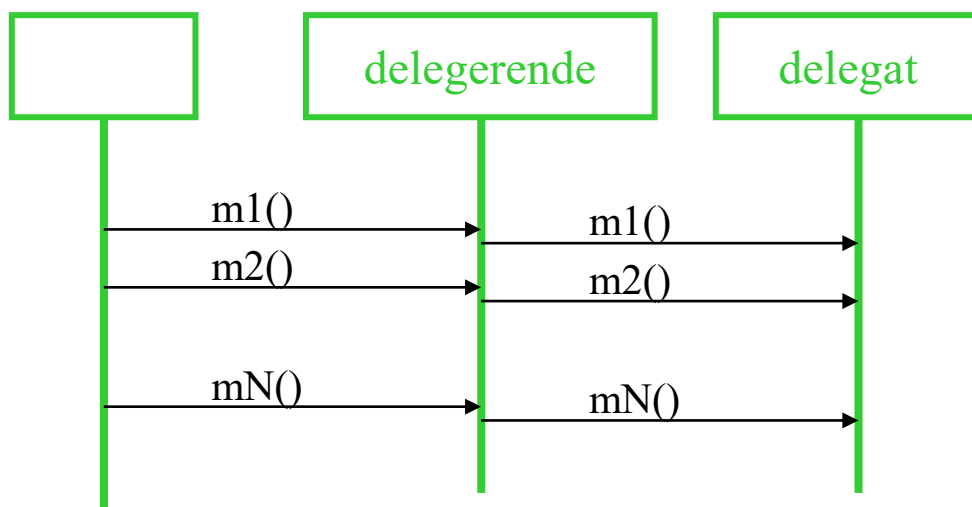
- getChildCount -> size
- getChild -> get
- addChild -> add
- removeChild -> remove
- ...



Se delegering/Person.java

Generell delegeringsløsning

- To objekter: delegerende og delegat
 - **delegerende** eier/har en referanse til **delegat**
 - **delegerende** har et sett metoder m1..mN som (noenlunde direkte) tilsvarer metoder hos **delegat**
 - kall til metodene m1..mN på **delegerende** fører (noenlunde direkte) til tilsvarende kall på **delegat**



Eksempel: IO

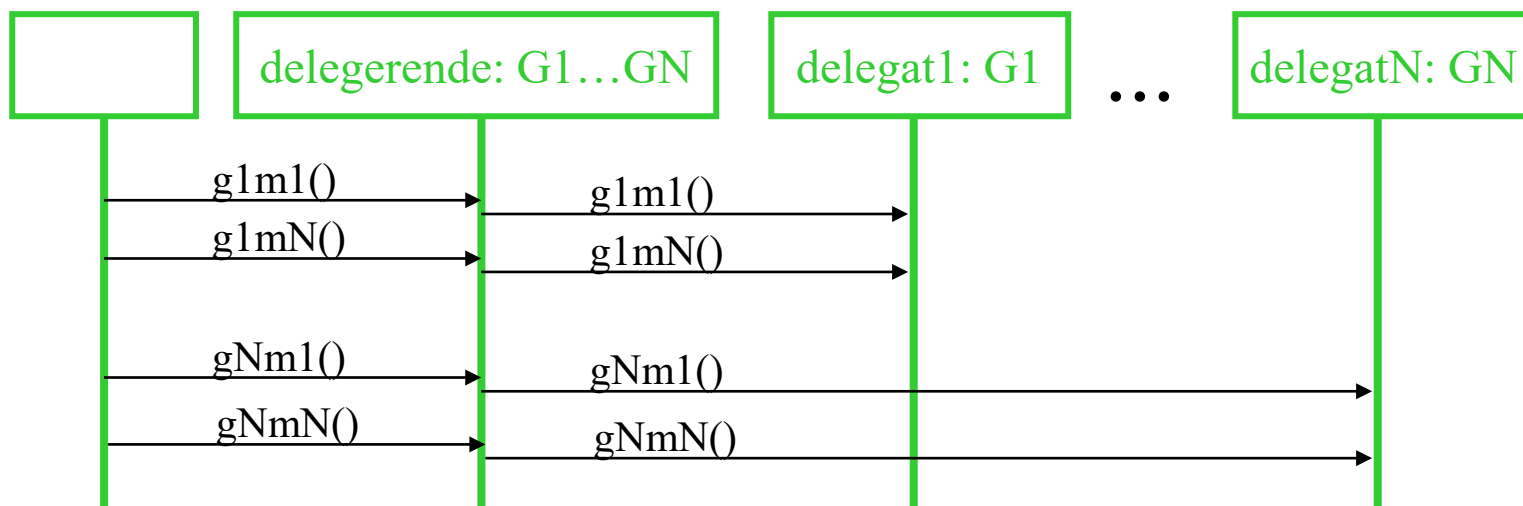
- Java sitt IO-system er bygd på delegering:
 - alle klassene bygger på en av klassene Writer, Reader, OutputStream og InputStream
- Mange av klassene delegerer til implementasjoner av disse, basert på riggingen av objektene med konstruktører
 - `new InputStreamReader(inputStream)`
ny `InputStreamReader` – delegerer til `inputStream`
 - `new BufferedReader(reader)`
ny `BufferedReader` – delegerer til `reader`
 - `new PrintWriter(writer)`
ny `PrintWriter` – delegerer til `writer`
 - ...
- Kall til metoder som `read/readln`, `write` og `print/println` delegerer til det indre objektet sine metoder

Noen eksempler

- Se kode i boss-mappen
- Se kode I kokk-mappen
- Merk at en ikke må definere interface for å kunne bruke delegering. Det er bare nyttig.

Delegering og grensesnitt

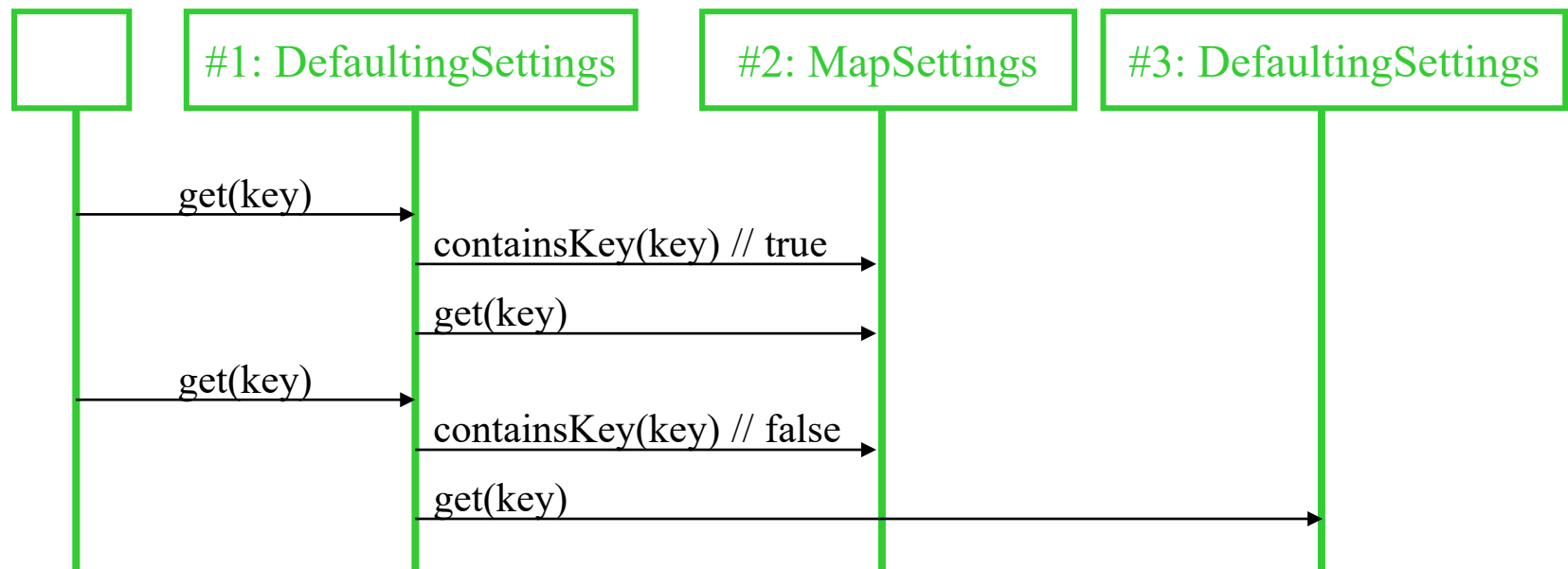
- Metodene $m_1 \dots m_N$ er ofte deklarerert i et grensesnitt som både **delegerende** og **delegat** implementerer
- **Delegerende** kan implementere flere slike grensesnitt og har da én **delegat** for hvert



- Dette er en form for *komposisjon/aggregering*: helheten er summen av egenskapene til delene

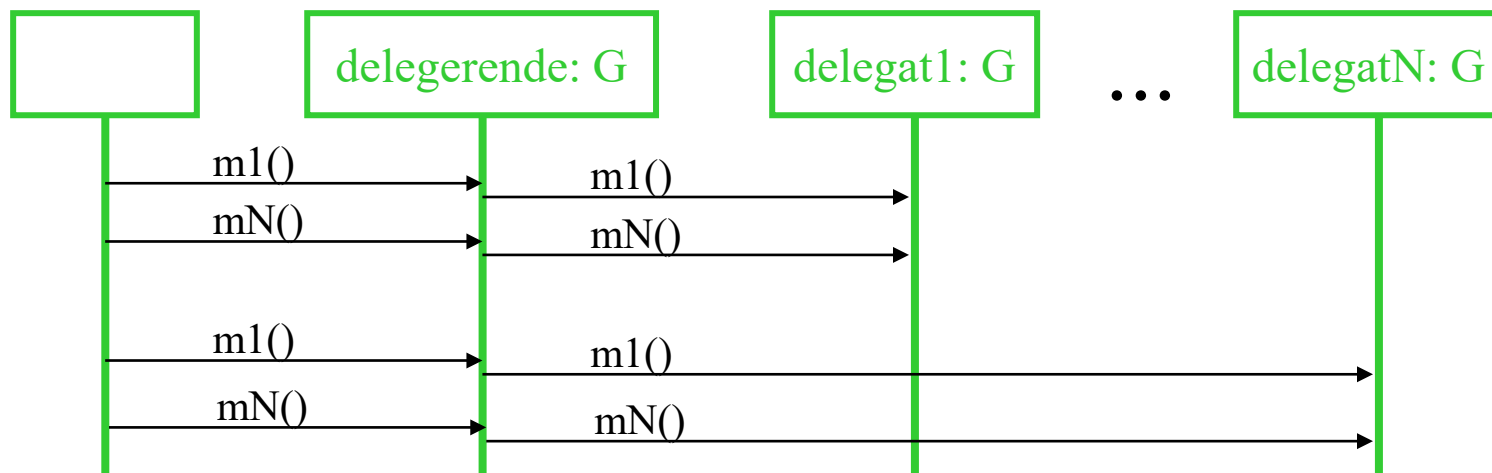
Eksempel: Settings med defaults

- En begrenset Map-aktig klasse for hierarkiske innstillinger/kjede med defaults.



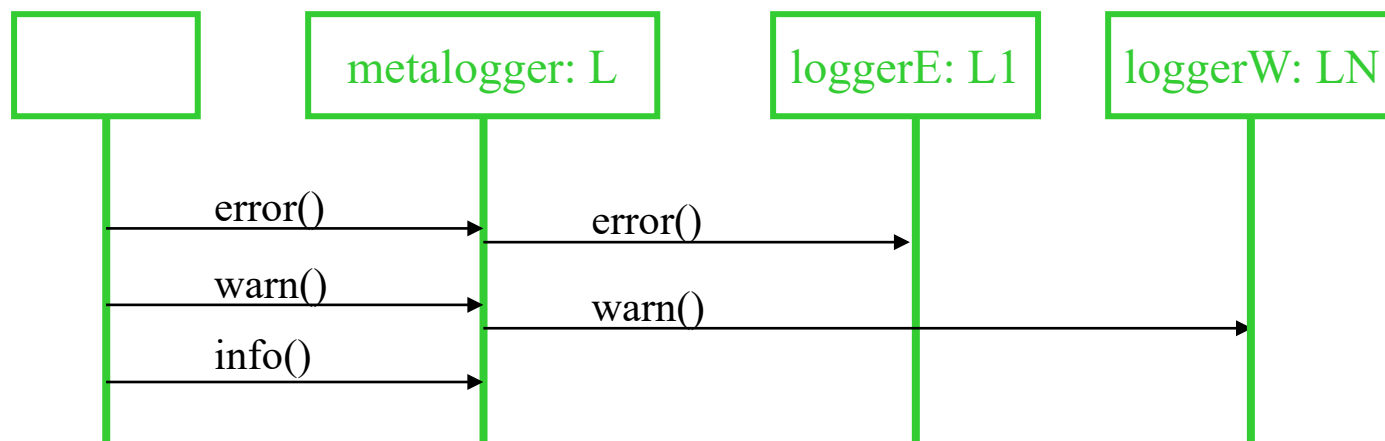
Varianter av delegering

- Flere **delegater** implementerer samme grensesnitt
- **Delegerende** velger hvilken det delegeres til i hvert tilfelle, basert på ikke-funksjonelle egenskaper
 - én implementasjon er raskere for bestemte argumenter
 - jevn fordeling av last på underliggende maskinvare
 - konfigurasjon som avhenger av formål, f.eks. drift, testing
- Delegerende kan tilpasse og filtrere metodekallene



Eksempel: logging

- Logging brukes for å samle informasjon om kjøring av program
- Mange rammeverk for logging, med noenlunde samme funksjonalitet
 - registrering av div. info om kilden: klasse, tidspunkt, ...
 - meldinger med ulik alvorlighetsgrad: **fatal**, **error**, **warning** og **info**
 - ulike måter å rapportere på: til fil, meldingspanel, rss, twitter, ...
 - regler for hvilke meldinger som skal havne hvor, f.eks. kaste **info**, lagre **warning** til fil, sende **error** til admin og **fatal** til brannvesenet
- Såkalte meta-loggere lar deg programmere med ett rammeverk, men delegere til et eller flere andre



- Hva med arving? Hvordan og hvorfor ikke?

Logging-oppgave, øving 6

[https://www.ntnu.no/wiki/display/tdt4100/
Delegering+-+Logger-oppgave](https://www.ntnu.no/wiki/display/tdt4100/Delegering+-+Logger-oppgave)

Kort om arv (igjen)



Arv gir muligheten til definere nye klasser og grensesnitt fra andre klasser eller grensesnitt.

Subklassene arver alle felt og metoder fra *superklassen*, men hvor vi kan

- **redefinere** metoder (vi har allerede sett @Override når vi f.eks. redefinerer toString())
- utvide med nye felt og metoder.
- Variable deklartert som type av superklasse, kan referere til objekter av alle subklasse-typer (som for grensesnitt)

Se: kortomarv, Account og CreditAccount.

Arv (eng: inheritance)



- instanser av subklassen vil inneholde alle felt og metoder (inkl. konstruktører) deklartert i både superklassen og subklassen
- alle felt og metoder som er deklartert som **public**, default (package) og **protected** i superklassen, er tilgjengelig i subklassen.

Arv gjør at vi kan...



- **Gjenbruke kode.** Vi skrive bare kode for det som vi vil *endre og legge til*.
- Bruke objekter av subklasser der hvor superklassen brukes.
- Konseptuelt organisere *klassene etter fellestrekk*. Det kan hjelpe i modelleringen.

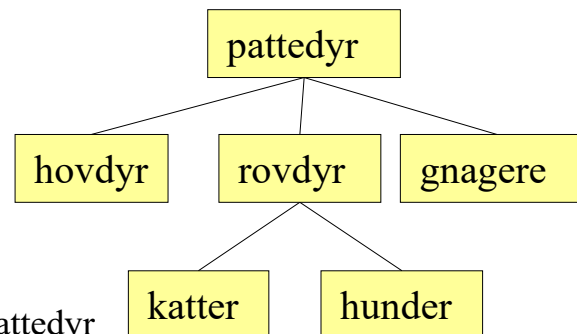
NB! Dette er på klassenivå, og ikke som assosiasjoner på objekt-nivå som vi hadde i Folder-eksemplet.

Hvorfor arv?



- Klassifisering i klassehierarkier er en intuitiv organisering av fenomener i verden

- hovdyr er et pattedyr
- katt og hund er et rovdyr er et pattedyr
- gnager er et pattedyr



- Arv gir mulighet for gjenbruk av kode
- Forenkler strukturering av komplekse problemer og programmer

Arv eller ikke?

- Det kan være fristende å bruke arv fordi vi slipper å skrive så mye kode, men:
- Arv gir en rigid, sterk kobling mellom klasser/objekter, som kan gjøre endringer mer komplisert å gjennomføre
- En kan ikke arve fra flere en klasse (direkte)
- **Liskovs substitusjonsprinsipp**: Kode som er korrekt, forblir korrekt hvis subklasse-objekt erstatter superklasse-objekt.
 - Her menes at programmet utfører det det er ment korrekt. Det er ikke nok at koden kompilerer.

Delegering vs. arv

Delegering

- Når du ønsker fokus på funksjonalitet (f.eks. skjule unødvendige funksjoner ved arv)
- Når du har behov for å kombinere funksjonalitet (dvs. komposisjon)
- Når arv blir helt feil (f.eks. *Car extends ArrayList<CarComponent>* hvor *ArrayList<CarComponent>* er tenke å brukes for bilens bestanddeler)

Arv

- Når du ønsker å dele basisfunksjonalitet (f.eks. *Dog extends Animal* hvor funksjoner for *Animal* er like relevant for *Dog*)
- Når du har behov for polymorfisme (f.eks. *Animal* som felles referanse og subklasse *Dog*, *Cat* osv. detekteres ved kjøretid). Dette får vi også ved bruk av grensesnitt.

Generelt råd: Velg komposisjon fremfor arv.

Læringsmål for forelesningen

- Objektorientering
 - Delegeringsteknikken

