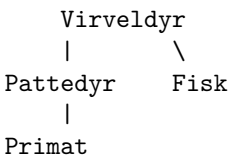
Repetisjonskurs: Arv, subklasser og grensesnitt

- Enkel arv
  - Subklasser
  - Referanser
  - Abstrakte klasser
- Polymorfi
- Grensesnitt
- Multiple inheritance

Enkel arv

- En subklasse arver foreldreklassens egenskaper og kan legge til flere, fungerer slik som en spesialisering.
  - For eksempel: Et pattedyr kan spise mat. En ulv kan i tillegg hyle og har fire bein, mens en ape kan brøle og har to bein.
- Eksempel på familietre av dyr se denne øvingsoppgaven



- Syntaks: Class B extends A {}
- Object: Klassen alle klasser arver fra implisitt
  - Dette kan vi se i Java-dokumentasjonen
  - Se på toString() og equals()
- Abstrakte klasser
  - Dersom en klasse bare skal fungere som et overhengende rammeverk for sine subklasser kan vi gjøre den abstrakt.
    - For eksempel: Pattedyr
    - En abstrakt klasse kan aldri instansieres, det vil si at vi ikke kan få et Pattedyr-objekt, men må velge en subklasse.
- Tilgang:
  - private: Kun denne klassen har tilgang
  - protected: Subklasser og pakke har tilgang
  - public: Alle har tilgang
- Datastrukturtegning:

  - Se datastruktur-notatet
  - En subklasse inneholder alt superklassen inneholder og så sine spesialiserte egenskaper
  - Kan tegne opp spesialiserte egenskaper i en indre boks om man ønsker, men det er ikke nødvendig.

Polymorfi

- Overriding: Redefinerer metode fra en superklasse
  - For eksempel kan klassen Pattedyr ha metoden void lagLyd(). Klassen Primat kan overskrive denne til å skrive ut et brøl.
- Overloading: Samme metodenavn, annen metodesignatur
  - Metodesignatur: Består av navnet og parametertypene. Dersom parametertypene er annerledes er metodesignaturen annerledes, ergo har vi overloadet metoden.

Grensesnitt

- Litt som abstrakte klasser, men beskriver bare grensesnittet, det vil si "håndtakene" mot verden.
  - Metoder som spesifiseres i et interface er public by default.
- Vi spesifiserer kun hvilke metoder som må være med, ikke hvordan de implementeres. Eks:

```
public interface EtInterface {
    int metode(int a, int b);
    String metode2();
}
```

- Kan ikke inneholde variabler.
- Kan ikke instansieres som et objekt, men må implementeres av en klasse:

```
Class A implements EtInterface {}
```

- Et grensesnitt kan brukes som referanse på klassene som implementerer det, for eksempel kan vi si

```
EtInterface objektet = new A();
```

Multiple inheritance

- I Java har vi i utgangspunktet ikke lov til å arve fra flere klasser.
- Multippel arv kan føre til problemer. Eksempel: Deadly diamond of death
  - Vi tenker oss en klasse A med subklasser B og C, som igjen har D som felles subklasse (D arver altså fra både B og C).
  - Hvis klasse A har en metode, og klasse B og C begge har overskrevet denne, men D ikke har overskrevet den, må D arve metodeimplementasjonen. Men fra hvem?
- Grensesnitt kan hjelpe oss å løse dette problemet, da klassen kan få egenskaper fra både en superklasse og et eller flere grensesnitt, men grensesnittene tvinger ikke inn noen eksplisitt implementasjon.

Andre ting

- Referanser
  - Et objekt av klassen Pattedyr kan ha Virveldyr som referanse, men ikke motsatt
  - Spørsmål: Er dette tillatt?

    ```
    Fisk enFisk = new Virveldyr();
    Pattedyr etPattedyr = new Primat();
    ```

  - Når vi bruker en "generell" referanse (f.eks Pattedyr for Primat) vet ikke referansen om spesielle egenskaper i objektet. Vi må da "caste" tilbake, enten ved å lage en eksplisitt Primat-referanse eller slik:

    ```
    (Primat) etPattedyr.klatre();
    ```

    * Men vi bør være sikre på at objektet faktisk arver fra superklassen, ellers får vi feil!
- Vi kan sjekke om et objekt tilhører en klasse eller superklasse ved hjelp av instanceOf()
  - Men ofte vil vi heller bruke polymorfi (eks labyrint)

Live-kode

- Oppgaver:
  - Oppgave 2.2: Kodeanalyseoppgaver til arv
  - Oppgave 2.6: Overstyrende arvinger
  - Oppgave 3.1: Kjøtt- og planteetere
  - Oppgave 3.3: Miljødata for motoriserte kjøretøy