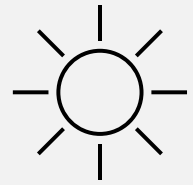
GRUPPE 11 IN1010 – UKE 4

~~Sivert Fjeldstad Madsen~~

Julian Fjeld

I DAG

- Repetisjon av arv
- Arv og komposisjon
- Polymorfi
- Konstruktører i subklasser
- Overloading
- Oppgaver



(KORT) REPETISJON AV ARV

- Lar oss modellere «familier» av objekter som deler enkelte egenskaper
- Gjør det mulig å representere data på en mer naturlig måte
- Viktig terminologi:
  - Subklasser og Superklasser
  - class Subklasse **extends** Superklasse {}
  - **instanceof**
  - **protected**

ARV VS KOMPOSISJON

- Komposisjon er når vi koder uten bruk av arv
  - Sånn som vi gjorde i hele IN1000
- Er ofte det enkleste og greieste når vi koder
- Arv egner seg godt når objektene vi skal kode har en «**er en**»-relasjon til andre objekter
  - En **hund** er et **dyr**
  - Et **kjøkken** er et **rom**
  - En **lenkeliste** er en liste
    - Mer om dette senere i kurset…

POLYMORFI

- Vi snakket litt om klassen Object forrige uke
- Alle klasser arver fra Object
- Tilbyr et par metoder som vi kan *redefinere*
  - F.eks toString() og equals()
- Det å redefinere metoder fra Superklasser i Subklasser kalles **polymorfi**
- Bruker annoteringen **@Override**

POLYMORFI - EKSEMPEL

- class Hund {
  - public void bjeff() {
    - System.out.println(«Voff!»);
  - }
- }

- Hund h = new Hund();
- h.bjeff();
  - -> Voff!

- class Puddel **extends** Hund {
  - **@Override**
  - public void bjeff() {
    - System.out.println(«Voff voff!»);
  - }
- }

- Puddel p = new Puddel();
- p.bjeff();
  - -> Voff voff!

KONSTRUKTØRER I SUBKLASSER

- Når vi bruker arv vil Java alltid prøve å kalle på Superklassens konstruktør når man oppretter et Subobjekt

- Vi må derfor eksplisitt fortelle Java hva den skal sende «opp» til klassen over
- Dette gjøres med nøkkelordet **super()**
  - Inne i parentesene skriver vi det vi vil «sende opp» til Superklassens konstruktør
- **super()** må alltid stå helt først i konstruktøren!
  - Ellers legger Java det inn automatisk, men da uten noen argumenter

SUPER - EKSEMPEL

- class Person {
  - int alder;

  - public Person(int alder) {
    - this.alder = alder;
  - }
- }

- class Student extends Person {
  - String universitet;

  - public Student(int alder, String universitet) {
    - super(alder);
    - this. universitet = universitet;
  - }
- }

OVERLOADING

- Overloading lar oss skrive den samme metoden flere ganger, men med forskjellig **signatur**

- Java skjønner da hvilken av metodene den skal bruke ut ifra hvilke argumenter som blir sendt inn
- Dette kan gjøre objektene vi skriver enda kraftigere og mer fleksible

OVERLOADING - EKSEMPEL

```
class Person {
    public void draPaaTur(int km) {
        System.out.println(«Personen drar paa en » + km + «km lang tur»);
    }

    public void draPaaTur(int km, String fremkomstmiddel) {
        System.out.println(«Personen drar paa en » + km + «km lang » + fremkomstmiddel + «tur»);
    }
}
```

OPPSUMMERING

- Med polymorfi blir objektene vi designer enda mer fleksible og kraftige
- Nøkkelordet **super()** gjør at vi kan ha ulike konstruktører i Super- og Subklasser
- Ofte er det greit å ikke bruke arv, og da kalles det komposisjon

- Ukas tips:
  - Lag tester til programmene dine
    - Gjerne aller først, før du begynner å kode noe annet!