NTNU

Kunnskap for en bedre verden

Øvingsforelesning 1

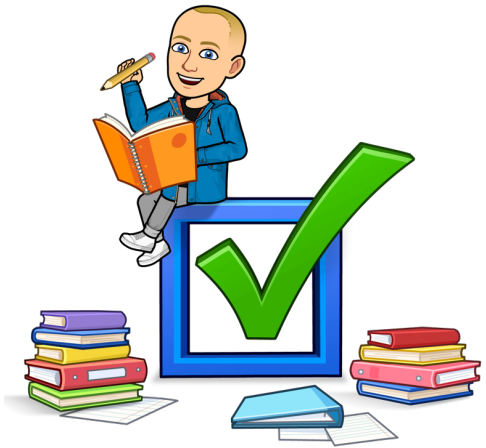
TDT4100 Objektorientert programmering

Eirik Lorgen Tanberg
Vitenskapelig assistent, TDT4100
eiritan@stud.ntnu.no

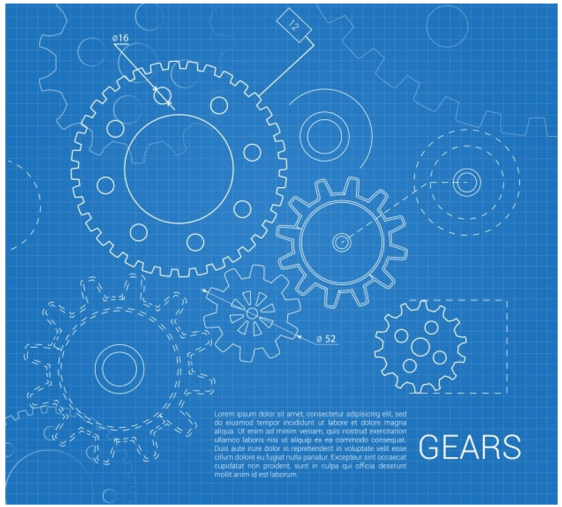
NTNU

Repetisjon av viktig teori

Objekter, klasser, terminologi, metoder

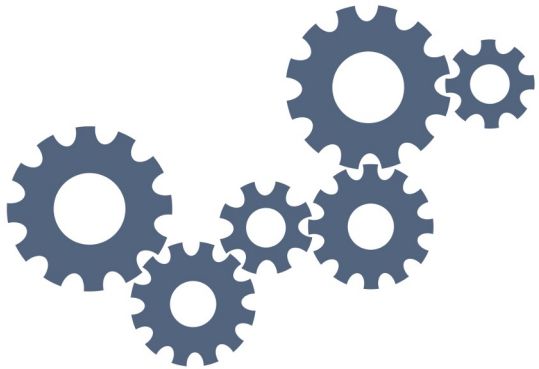


Klasser og objekter



Klasse

Instansiering



Objekt

Objekter

- Objekter er en del av et kjørende program som har
  - **Tilstand**
    - Data objektet **“husker”**
    - Lagres i variabler på samme måte som dere er kjent med fra ITGK. Disse variablene kalles **felt** i Java-terminologien
    - Tenk: Data som beskriver **dette ene** objektet
      - Evt: (se for deg data som beskriver spesifikt **deg** og ikke andre mennesker)
  - **Oppførsel**
    - Spørsmål du kan stille, tjenester du kan be objektet utføre
    - Oppførsel **endrer intern tilstand** over tid
- Fokuset i OOP er **oppførsel**, men tilstand og oppførsel utgjør en dualitet
  - Den ene kan sjelden eksistere uten den andre

Klasser

- En klasse, er en **mal** for hvordan et sett av objekter skal se ut. Den kan beskrives som en "tredelt boks":
  - **Navn**
    - (identitet) som identifiserer klassen
  - **Felt**
    - (variabler/attributter/egenskap) som inneholder tilstanden til klassen
  - **Metoder**
    - (oppførsel/operasjon) som inneholder den dynamiske oppførselen til klassen
- Klassen **innkapsler** data (tilstand) og oppførsel
- Den tredelte boksen kalles gjerne et klassediagram

Gjennomgang av terminologi

- **Deklarere**
  - Definere navnet og **datatypen** til en variabel
  - `int number;`
- **Initialisere**
  - Sette en verdi til en deklarert variabel for første gang
  - `String beskjed = "Hei! Husk å levere øving";`
- **Metode**
  - Navn for **funksjoner** innenfor objektorientert.
  - Generelt brukes begrepet metoder når de er definert som en del av en klasse (ligger inne i klassedefinisjonen)
- **Instans**
  - Et konkret “eksemplar” av noe
    - Et objekt er en instans av en klasse
    - Analogi: **Du** er en instans av “klassen” **Menneske**

Bestemme start-tilstand for objekt

- Er alle mennesker 100% like i begynnelsen?
  - Nei, man har f.eks. gener som definerer noen egenskaper ved seg
  - På samme måte må ikke alle nye objekter som instansieres av en klasse være “like” (i samme start-tilstand)
  - Vi kan definere parametere i **konstruktøren** til klassen som lar oss definere start-tilstanden

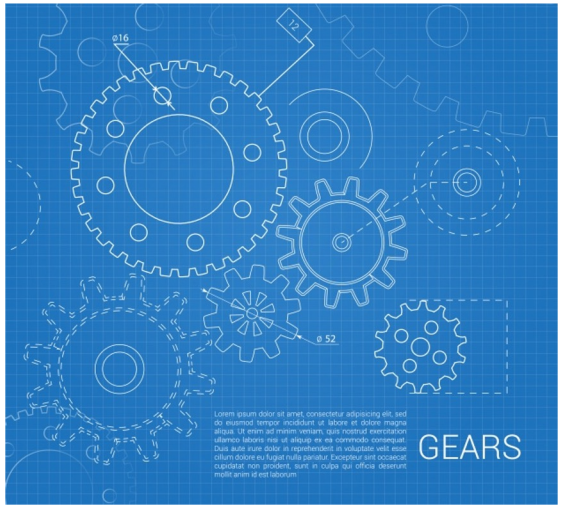
Konstruktør

- **Konstruktøren** er en funksjon som kjøres når et objekt **instansieres**:
  - `MinKlasse obj = new MinKlasse(<arguments>);`

- Navnet på «funksjonen» **må** være det samme som klassenavnet.
- Av konvensjon er det normalt å definere konstruktørene øverst i klassedefinisjonen, under feltene, men det er ikke et krav.
- Det er fullt mulig å ha flere konstruktører i en klasse, men disse må ha unike **signaturer** for å separere de
  - Konstruktørene må ha forskjellig input-parametere
  - En annen måte å separere konstruktører fra hverandre er ved å bruke forskjellige **synlighetsmodifikatorer** som vi kommer til neste uke.
- Objekter **må** ikke ha definert konstruktør, men da vil alle objekter starte i samme tilstand*

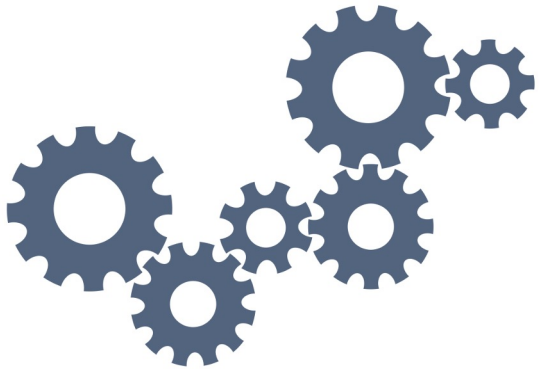
*Ikke alltid sant, men det kommer vi tilbake til

Klasser og objekter



Konstruktøren kjøres
under **instansiering**

```
Public Gears(...) {
    ...
}
```



Klasse **Instansiering** Objekt

Getters og setters

- Et sett med standardmetoder i klasser som dere vil bli **godt kjent** med fremover

- **Setters**
  - Overskriver/modifiserer et internt felt (variabel) i klassen
  - Lar oss definere *regler* for hvordan feltene kan endres
- **Getters**
  - Returnerer et internt felt (variabel) i klassen
  - Hjelper oss med å *skjule implementasjonsdetaljer*
- Mulig å autogenerere disse med tillegg hvis man ønsker

- Vi kommer tilbake til hvorfor vi må ha getters og setters neste uke
  - Det er ingenting “spesielt” med disse metodene, kun *konvensjon*

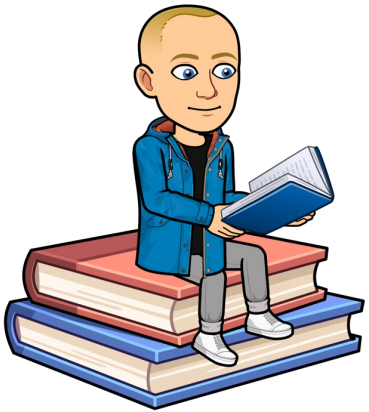
NTNU

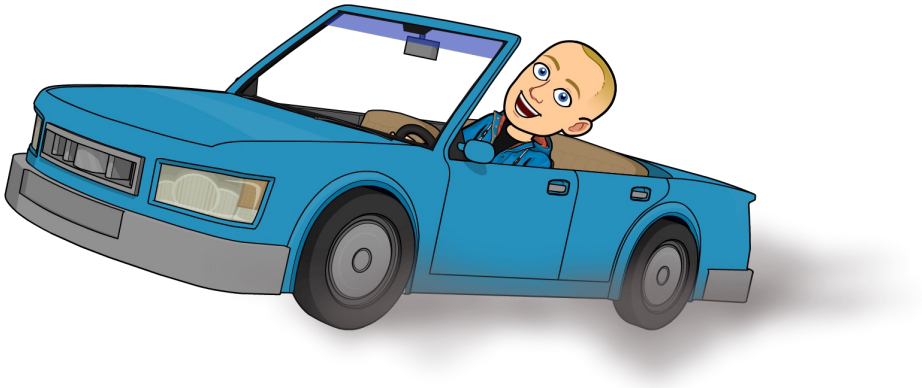
Oppgaveløsning

Book.java og Car.java

Case for oppgaveløsning

- Idag skal vi i hovedsak lage to forskjellige klasser:
  - **`Book`** og **`Car`**





Mappestruktur for ØF

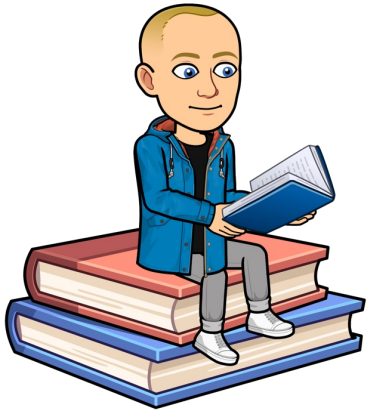
main/src/java

- of1
  - kode ← Her kan du kode direkte selv
  - If etterkant ← Løsningsforslag, legges ut i
- of2
- .
- .
- .
- of13

NTNU

Book.java

Klasser og objekter i praksis



Oppgave 1

I den tilhørende koden til denne forelesningen finnes det uferdig kode til en Book-klasse. Denne klassen er en enkel representasjon av en bok som holder styr på tittelen til boken og hvor mange sider den består av.

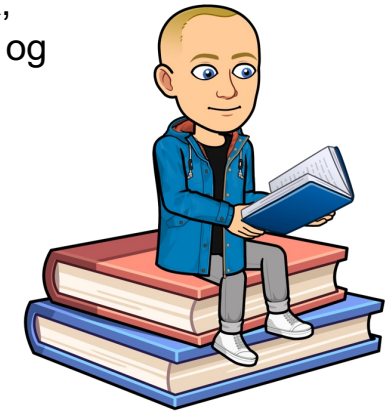
Legg til felter som representerer tittelen og antall sider i en bok. Implementer deretter kode for konstruktøren og getter- og setter-metodene slik at disse lagrer og gir ut korrekt informasjon.

For å teste at koden fungerer kan main-metoden kjøres. Denne lager en bok, prøver å sette en ny tittel og et nytt sideantall, samt henter og skriver ut tittel og antall sider. Ved korrekt implementasjon av Book-klassen, vil main-metoden skrive ut det følgende (cirka):

```
The book "Big Java" has 100 pages.

The book "Introduction to Algorithms" has 718 pages.

forelesning2.kode.Book@e9e54c2
```



`toString()`-metoden

- En **`toString()`**-metode er et **generelt konsept** men også en stadard måte å lage en **tekstrepresentasjon** av et objekt i Java.
- Generelt skal toString-metoden til et objekt returnere en **streng** som har en «tekstlig representasjon» av objektet. Dette bør være en kort, men informativ og lesbar representasjon av objektet.
- Alle objekter har **allerede** en toString-metode, men denne er ikke nødvendigvis så informativ uten at vi selv definerer hva som er en god representasjon av objektet som en tekststreng.
  - Må derfor **overskrive** standard toString med vår egen metode
- Eksempel:
  - Et **Person**-objekt (menneske) kan f.eks. representeres som

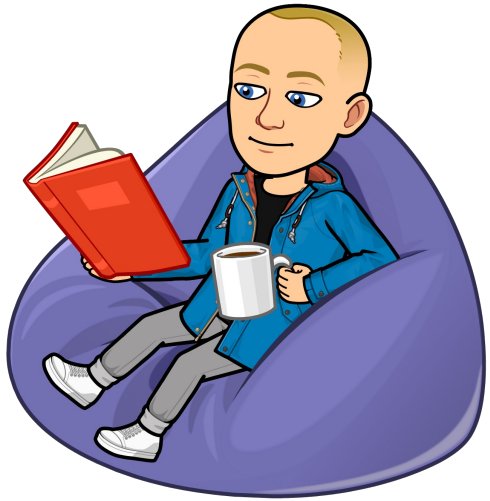
    *"Navn: **Eirik**, Alder: **22**, Kjønn: **Mann**"*

Oppgave 2

Siste linje i main metoden prøver å skrive ut **Book**-objektet, men det som skrives ut er ikke veldig nyttig. Dette er fordi **Book** ikke har noen **`#toString()`**-metode.

Lag en beskrivende `#toString()`-metode for Book

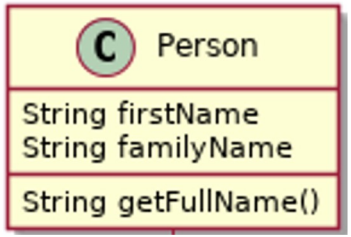
```
The book "Big Java" has 100 pages.
The book "Introduction to Algorithms" has 718 pages.
forelesning2.kode.Book@e9e54c2
```



Visualisering av klasser og objekter

- En del av pensum i emnet er å kunne visualisere klasser/objekter/tilstander++

- Vi visualiserer klasser med **klassediagram**:

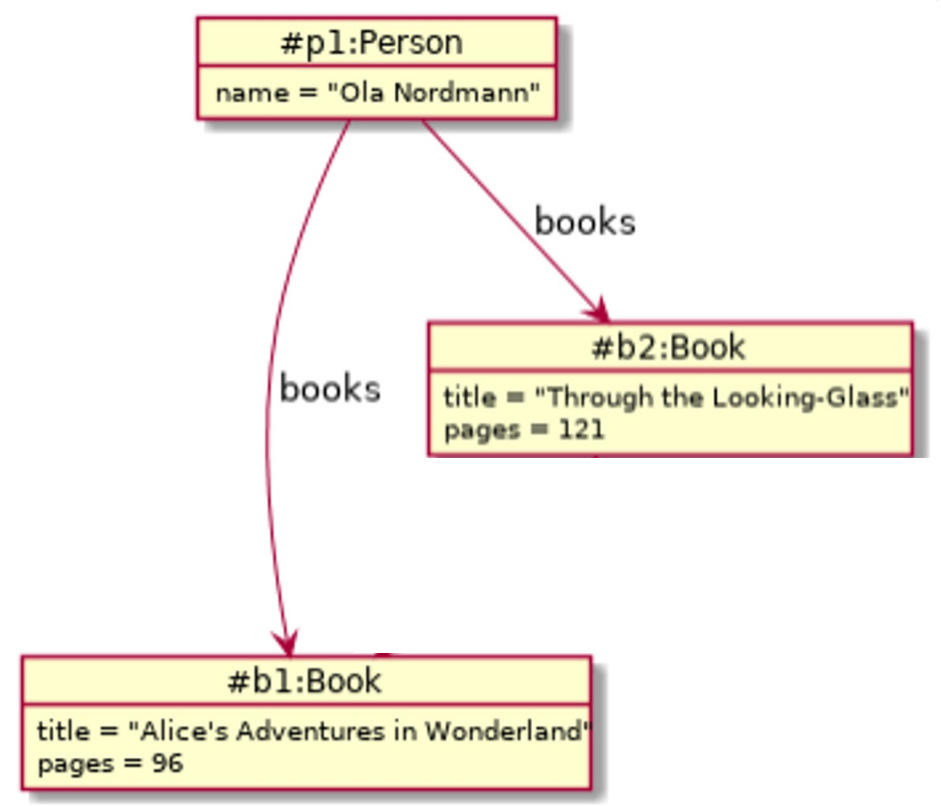


Objektdiagrammer

*Distinkte instanser av menneske: deg og meg

- Vi visualiserer **distinkte instanser*** av klassene og hvordan de henger sammen

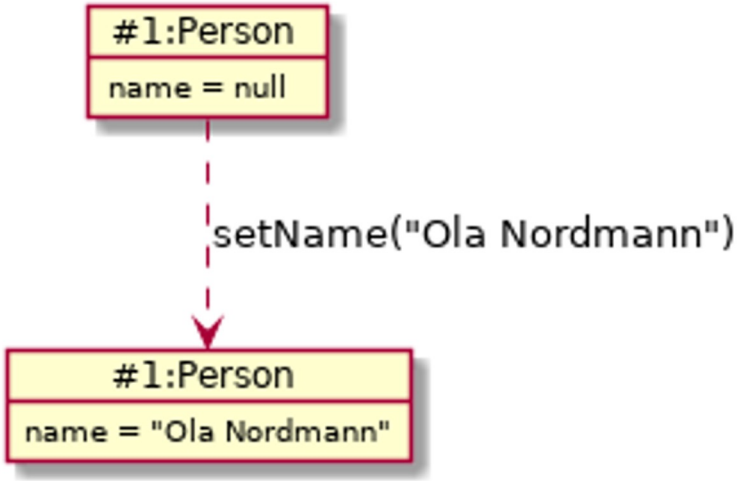
- Viser intern tilstand med verdier i feltene, og referanser til andre klasser



Objekttilstandsdiagram

- Brukes for å visualisere **kombinasjonen** av tilstand og oppførsel, dvs. hvordan de ulike operasjonene endrer tilstanden på et objekt

- Veldig relevant for øving 1



Generelt om diagrammer

- Dere finner det meste dere trenger av informasjon og pensum om diagrammer på wikien:
  - https://www.ntnu.no/wiki/display/tdt4100/Diagrammer

- Det er ikke veldig farlig hvordan dere tegner diagrammer, det holder å skrive i notatblokk og digitalisere dette.
  - diagrams.net er et godt og gratis verktøy som brukes av veldig mange

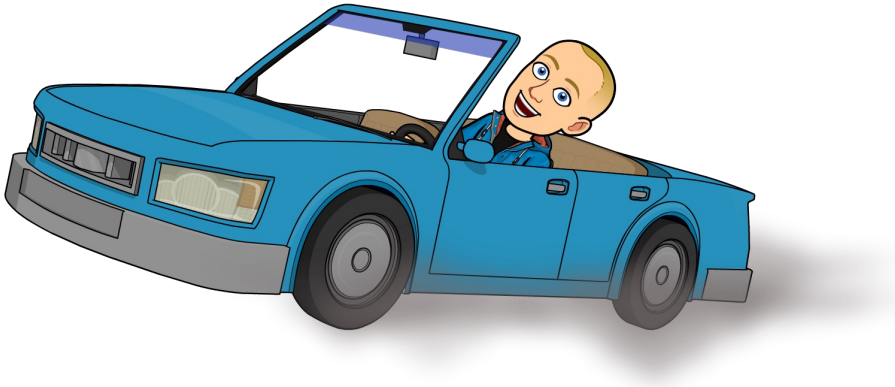
Oppgave 3

- Lag et objekttilstandsdiagram for **Book**-klassen for følgende operasjoner etter hverandre:
    - Et book-objekt er opprettet med 100 sider og tittel Big Java
    - Hent ut NumPages
    - Sett NumPages lik 99
    - Hente ut NumPages
    - Sett tittelen til boken lik “Algorithms”
    - Hent ut tittelen
    - Print objektet

NTNU

Car.java

Mer kompliserte objekter



Oppgave 4

Vi ønsker å lage en klasse som representerer en bil. Bilen skal ha **merke**, **modell**, **årstall** og **antall kilometer kjørt**. Alle feltene skal ha getters og setters der det er naturlig. Lag også en **`#toString()`**-metode.

Oppgave 5

En bil står ikke bare stille. Hele poenget med en bil er å kunne kjøre rundt. Vi ønsker å kunne representere om en gitt bil står stille eller om den er i bevegelse. Dette kan vi for eksempel gjøre med å ta vare på farten til en bil. Når man lager en ny bil står denne som regel stille, så farten kan settes til 0 til å starte med. For å endre på farten kan man for eksempel lage metoder for akselerasjon og bremsing.

Lag en variabel for å holde styr på farten til en bil og en getter-metode for denne. Lag også metoder for akselerering og bremsing som tar inn hvor mye farten henholdsvis øker eller minskes. Ingen av disse metodene skal endre farten hvis den gitte endringen er negativ.

Pass på at farten ikke kan bli mindre enn 0 ved bremsing.

Oppgave 6

Det gir ikke mening å la det være mulig å sette kilometerstanden til en bil direkte, da denne bare skal kunne **øke**. I stedet bør det finnes en metode som øker kilometerstanden med et gitt antall kilometer.

Lag en metode som øker kilometerstanden med et gitt antall kilometer. Hvis et negativt antall kilometer blir gitt inn, skal ikke kilometerstanden endre seg.

Å komme seg over kneika

- TDT4100 er en **stor** overgang fra ITGK, og vi vet at dette stresser mange.
- Det er **ikke** forventet at dere kan all syntaks og alle konsepter de første ukene, selv om vi benytter oss av noe av det i forelesning.
- Fordi OOP er mer sammensatt enn ITGK og det meste bygger på hverandre, så er vi litt avhengig av å introdusere en del konsepter nå, men **ikke bli stresset om du ikke skjønner alt** - vi skal repetere og gå gjennom det mange ganger!
- Det er mange som har gjort det bra i faget, selv om de første ukene er forvirrende. **Ta det med ro, og ta et steg av gangen**.

Lykke til med ukas øving!

Spørsmål og tilbakemeldinger kan sendes til
eiritan@stud.ntnu.no