

# Vi fortsetter igjen 11:15

**Forelesningen tas  
opp automatisk mens  
det røde lyset ved  
kateteret er på. 🎥**





NTNU

Kunnskap for en bedre verden

# Forelesningen begynner 10:15

**Forelesningen tas  
opp automatisk mens  
det røde lyset ved  
kateteret er på. 🎥**

TDT4100 Objektorientert programmering | 16.02.2024





NTNU

Kunnskap for en bedre verden

# Øvingsforelesning 4

TDT4100 Objektorientert programmering

16.02.2024

**Mathea Berg Vindsetmo**

Vitenskapelig assistent, TDT4100

[matheabv@stud.ntnu.no](mailto:matheabv@stud.ntnu.no)



# Agenda

- Administrativt
- Introduksjon til øving 4
- Teori for øving 4
  - Testing
  - Assosiasjoner
- Praktisk oppgaveløsning




# Administrativt

-

# Unngå vanlige VSCode-feil

- Ikke bruk **mellomrom** i mappenavn hvor du har repoet til TDT4100
- På Mac: Ikke plasser prosjektet i en mappe som er **synkronisert med iCloud** (desktop, documents, etc.)



```
PROBLEMS 1K+ OUTPUT DEBUG CONSOLE TERMINAL Filter (e.g. text, !exclude) Launch CardDeck
Error occurred during initialization of boot layer
java.lang.module.FindException: Error reading module: /Users/ /Documents/2. NTNU/Java/students/ovinger/target/classes
Caused by: java.lang.module.InvalidModuleDescriptorException: module-info 2.class found in top-level directory (unnamed package not allowed in module)
```

# JavaFX for M1 Mac

- Noen mulige løsninger (for både JavaFX og SceneBuilder) ute på wiki nå:
  - <https://s.ntnu.no/tdt4100-m1-fix>
- Prøv gjerne ila. forelesningen og gi tilbakemelding om det fungerer for deg eller ikke.



# Introduksjon til Øving 4

Læringsmål og informasjon



# Øving 4

- Tema: **Objektstrukturer**
- Øvingsmål
  - Lære hva assosiasjoner er og hvordan dette brukes i OOP
  - Lære hvordan man sikrer konsistent oppførsel mellom assosierte objekter
- Øvingskrav
  - Kunne implementere klasser som har assosiasjoner til én eller flere andre klasser
  - Kunne sikre at disse assosiasjon er konsistente i enkle objektstrukturer
  - Kunne implementere metoder som oppretter, oppdaterer og fjerner slike assosiasjoner

# Øving 4

- Denne øvingen er delt opp i tre forskjellige deler:
  - Del 1:
    - Vanlig oppgaveløsning, må enten gjøre
      - to *lette* oppgaver + en litt lengre oppgave
      - to litt lengre oppgaver
  - Del 2:
    - Lag *klassediagram* for oppgaven du velger
      - NB! **Card** må tegnes hvis du velger denne

# Teori

Testing og assosiasjoner

# Assosiasjoner

Eksempel på assosiasjon:

- Klassen **Filmvisning** inneholder et felt av typen **Film**

Det finnes flere typer assosiasjoner:

- 1-1
- 1-n
- n-n


```
public class Filmvisning {  
    String sal;  
    String tid;  
    int pris;  
    Film film;  
}
```

# Assosiasjoner

1 - 1	1 - n	n-n
<b>Kjærester</b> 	<b>Mor med barn</b> 	<b>Venner</b> 

# 1-1 Assosiasjon


```
public class Filmvisning {  
  
    Sal sal;  
    String tid;  
    int pris;  
    Film film;  
}
```



```
public class Film {  
  
    String name;  
    int alder;  
    String nasjonalitet;  
}
```

# 1-n Assosiasjon

```
public class Bokhylle {  
    List<Bok> samling = new ArrayList<>();
```



```
public class Bok {  
    private String tittel;  
    private int sider;  
    private int hvor;
```

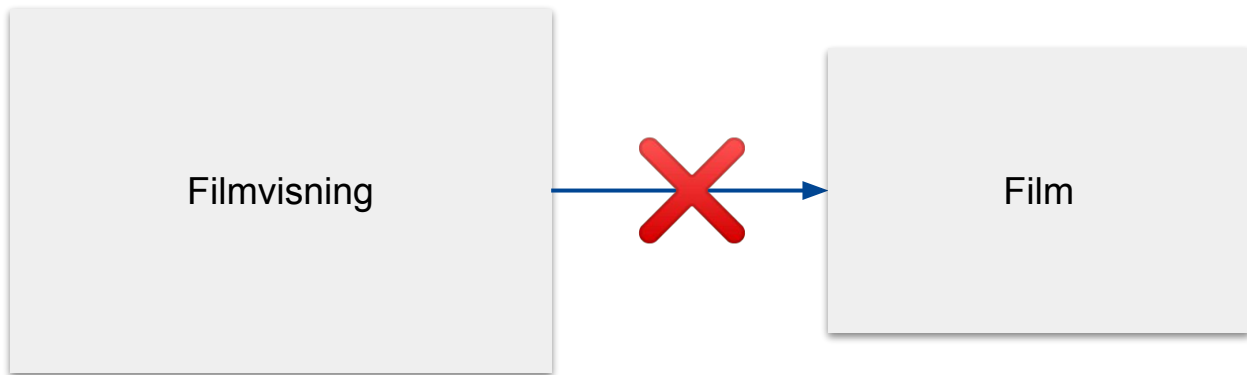
# n-n Assosiasjon

```
public class Emne {  
    String kode;  
    Collection<Person> forelesere;  
}
```

```
public class Person {  
    String navn;  
    Collection<Emne> foreleser;  
}
```

# Assosiasjoner og null

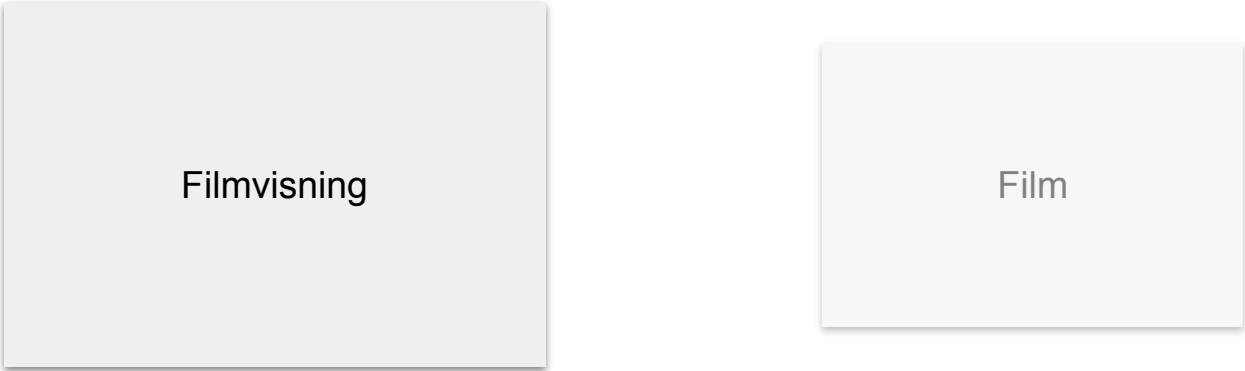
- Alle typer felt/variabler kan settes til **null**
- Felt i et objekt kan settes til **null** for å “bryte” assosiasjonen (koblingen) mellom det og et annet objekt



```
Film film = null;
```

# Assosiasjoner og null

- Alle typer felt/variabler kan settes til **null**
- Felt i et objekt kan settes til **null** for å “bryte” assosiasjonen (koblingen) mellom det og et annet objekt



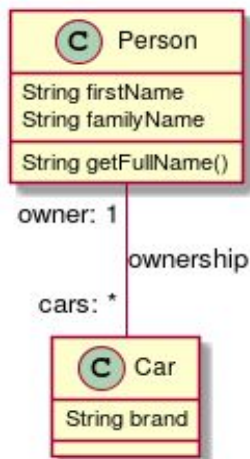
Filmvisning

Film

```
Film film = null;
```

# Assosiasjoner i klassediagram

- Felter som representerer assosiasjoner til andre objekter skal vanligvis ikke listes opp med resten av feltene i et klassediagram, men heller representeres som en **kobling** mellom to klasser:

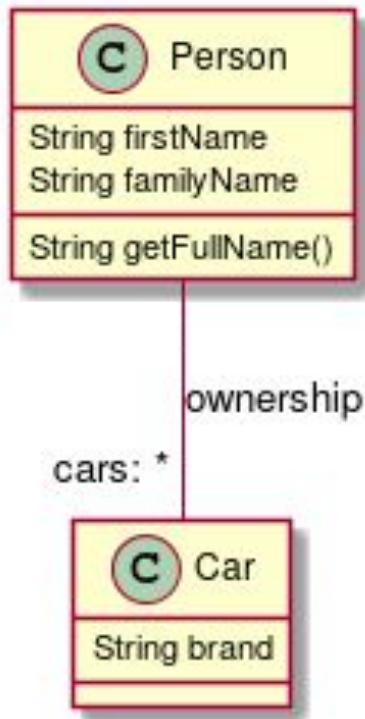


```
class Person {
    String firstName;
    String familyName;
    String getFullName();
    Collection<Car> cars;
}

class Car {
    String brand;
    Person owner;
}
```

# Toveis assosiasjon

- Når begge klasser har **assosiasjon** til hverandre i koden

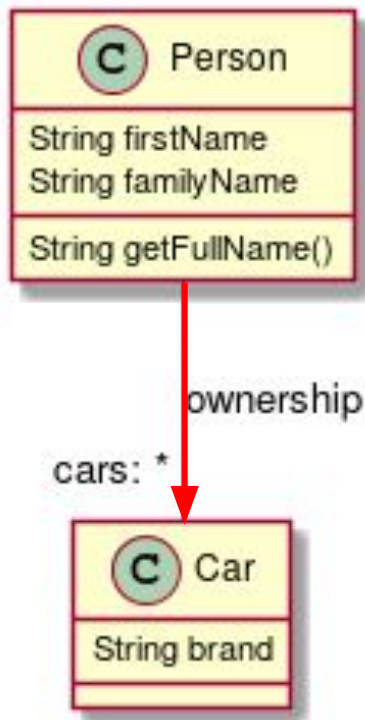


```
class Person {
    String firstName;
    String familyName;
    String getFullName();
    Collection<Car> cars;
}

class Car {
    String brand;
    Person owner;
}
```

# Enveis assosiasjon

- Hvis en assosiasjon har et pilhode, så betyr det at den er enveis



```

class Person {
    String firstName;
    String familyName;
    String getFullName();
    Collection<Car> cars;
}

class Car {
    String brand;
}
  
```

Merk at her kan vi *navigere* fra **Person** til **Car**, men ikke fra **Car** til **Person**

# Praktisk oppgaveløsning

Vi tar en liten pause fra *OOP mini* sine selvbetjente kassaapparat og hjelper de i stedet med å modellere konsernstrukturen deres



# Oppgave 1

Det første vi skal gjøre er å representere generelle bedrifter og tilhørende, generelle, underavdelinger. For å gjøre jobben vår litt enklere, vil vi representere en bedrift som en avdeling *som ikke har noen av avdelinger over seg*. Dette vil si at avdelinger og bedrifter representeres av *den samme klassen*.

Lag en klasse ***Department*** med tilhørende konstruktører. ***Department*** skal ha en ***assosiasjon\**** til avdelingen den tilhører, og man må ha en ***getter*** til denne. Det skal være mulig å initialisere både med og uten å oppgi hvilken overordnet avdeling den tilhører.

*\*Dette vil si at den må ha et internt felt som er et annet Department objekt.*

# Oppgave 2

En avdeling trenger ikke bare å vite hvilken avdeling som er overordnet en spesifikk avdeling, men også hvilke eventuelle underavdelinger som tilhører den igjen.

**Legg til et internt felt bestående av *Department*-objekter i *Department*. Lag deretter en privat metode *addSubDepartment*, som tar inn et *Department*-objekt og legger til dette i listen. Denne metoden skal kalles i konstruktøren som tar inn en overordnet avdeling med objektet som kaller på metoden i seg selv som input-argument**

*Merk: Vi bør også sjekke om vi **allerede** har lagt til den samme underavdelingen før vi legger den til i listen*

# Assosiasjoner og this



Child



Mother



this

# Assosiasjoner og this

Child



Mother



```
this.getMother();
```

# Assosiasjoner og this



Child



Mother



```
this.getMother().getChild();
```

# Oppgave 3

I tillegg til at en avdeling har oversikt selv over hvilke underavdelinger den har selv, bør vi også kunne sjekke dette via en metode.

**Skriv en (public) metode *containsDep* i *Department* som tar inn en avdeling og sjekker om den er en underavdeling av avdelingen som *contains* ble kalt på.**

*Merk: Her kan vi se for oss mange “nivå” med underavdelinger, men for å begrense kompleksiteten (og unngå bruk av rekursjon) vil vi her kun tenke på **direkte underavdelinger og underavdelinger av underavdelinger.***

# Oppgave 4

Vi er klare til å begynne å legge til ansatte i OOP-konsernet nå, som skal kunne høre til (via *assosiasjon*) en av avdelingene.

**Lag en klasse *Employee* med tilhørende konstruktør. En ansatt skal vite hvilken avdeling han/hun tilhører (internt felt) og få dette oppgitt i konstruktøren.**

**Legg deretter til en liste med *Employee*-objekter i *Department***

# Oppgave 5

Det er ikke veldig nyttig at **bare** den ansatte vet hvilken avdeling han/hun tilhører, så som de fleste andre konsern bør også ha en oversikt over hvilke ansatte som er tilknyttet.

Utvid *Department*-klassen med metoden *addEmployee* som tar vare på en liste med ansatte som tilhører avdelingen. Denne metoden skal kalles på *Department*-objekter som settes i konstruktøren til *Employee*. Husk å sende med det aktuelle *Employee*-objektet som input-parameter i *addEmployee*.

Lag deretter til slutt også en *removeEmployee*-metode som fjerner en ansatt fra en avdeling.

# Oppgave 6 a)

Ansatte kan få forfremmelser. De vil da bli flyttet til avdelingen **over** den de allerede er på.

**Lag en metode *promote* i *Employee* som forfremmer den ansatte hvis mulig, ellers skal det kastes et *IllegalStateException*.**

Oppskrift:

- Assosiasjonen mellom den ansatte og avdelingen må brytes
- Det må opprettes en ny assosiasjon mellom den ansatte og den overordnede avdelingen til avdelingen til den de tidligere var ansatt ved.

## Oppgave 6 b)

En ansatt bør også kunne si opp stillingen sin og dermed ikke tilhøre en spesifikk avdeling.

**Lag en metode *quit* i *Employee* som bryter assosiasjonen mellom den ansatte og dens tilhørende avdeling. Dersom den ansatte ikke tilhører noen avdeling skal det kastes en *IllegalStateException*.**

# Oppgave 7

Vi lager et forklarende **klassediagram** som viser oppbyggingen til klassene **Department** og **Employee** og samspillet mellom disse.



# Lykke til med ukas øving!

Spørsmål og tilbakemeldinger kan sendes til  
[matheabv@stud.ntnu.no](mailto:matheabv@stud.ntnu.no)