# Fra krav til objekter

**Ansvarsdrevet OO:**

**CRC og UML Sekvensdiagrammer**

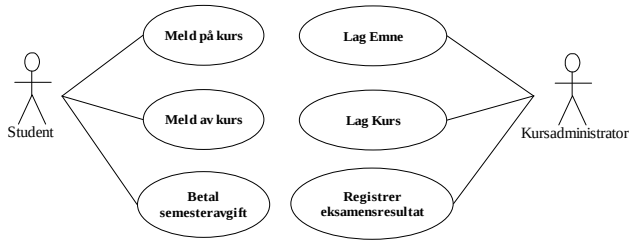


# Dagens forelesning

- **Kort repetisjon av kravspesifikasjon med UML**
  - **Hva skal systemet gjøre?**
  - **UML: Bruksmønstermodeller (Use Cases)**
- **Objektdesign**
  - **Hvordan skal systemet fungere?**
  - **Tre typer objekter**
  - **CRC: Hvordan finne "gode" objekter?**
  - **UML: Sekvensdiagrammer**



# Kursregistrering bruksmønstermodell




**Skiller mellom "Emne" (f.eks. Inf1050) og "Kurs" i emnet (Inf1050 v08)**

**Variasjon for "Lag Emne": Opprett nytt emne**

**Variasjon for "Lag Kurs": Opprett nytt kurs**

**Variasjoner for "Meld på kurs":**
**Kurset forutsetter andre emner: Studenten må ha bestått kurs for emnene**
**Dersom et kurs er fullt: Studenten kan bli satt på venteliste**

**Variasjon for "Meld av kurs":**
**Dersom kurset var fullt: Første student på venteliste blir meldt på kurset**



# Spesifikasjon av "Lag emne"

**Navn:** Lag emne

**Aktør:** Kursadministrator

**Hovedflyt:**

1. Kursadministrator velger emnekode
2. Systemet finner emnet
3. Kursadministratoren oppdaterer beskrivelsen av emnet (her kan det være mange underpunkter og variasjoner, for eksempel registrering av hvilke andre emner som forutsettes)
4. Systemet registrerer den nye informasjonen

…

**Alternativ flyt, steg 2:** Emnekoden eksisterer ikke:

A.1.1. Systemet spør om nytt emne skal opprettes og oppretter i så fall nytt emne med gitt emnekode (dvs, "Nytt emne" er en variasjon over "Lag emne")

**Relatert informasjon:**

Pga behov for historikk og avhengighet til kurs kan man ikke slette emner, men det bør være mulig å definere at et nytt emne *erstatter* et gammelt emne under punkt 3.

# Tekstlig spesifikasjon av ”Meld på kurs”

**Navn:** Meld på kurs

**Aktør:** Student

**Prebetingelse:** Student har betalt semesteravgift

**Postbetingelse:** Student er meldt på kurset eller er satt på venteliste

**Hovedflyt:**

1. Studenten velger emne
2. Systemet sjekker at studenten er kvalifisert til å ta emnet
3. Systemet finner kurs for emnet
4. Systemet sjekker om det er ledig plass på kurset
5. Systemet registrerer studenten på kurset



# ”Meld på kurs” (forts.)

Alternativ flyt, steg 1: Emnet finnes ikke:

A.1.1 Bruksmønsteret avsluttes

Alternativ flyt, steg 2: Emnet forutsetter andre emner:

A.2.1 Systemet sjekker at studenten har bestått kurs for emner som forutsettes

Alternativ flyt, steg A.2.1: Studenten har bestått kurs for emner som forutsettes:

A.2.1.1.1 Bruksmønsteret fortsetter fra steg 3

Alternativ flyt, steg A.2.1: Studenten har ikke bestått kurs for emner som forutsettes:

A.2.1.2.1 Bruksmønsteret avsluttes

Alternativ flyt, steg 3: Det holdes ikke kurs i emnet dette semesteret:

A.3.1 Bruksmønsteret avsluttes

Alternativ flyt, steg 4: Kurset er fullt:

A.4.1 Systemet spør om studenten ønsker å bli satt på venteliste

Alternativ flyt, steg A.4.1: Studenten ønsker å bli satt på venteliste:

A.4.1.1.1 Systemet setter studenten på venteliste.

A.4.2 Bruksmønsteret avsluttes



# Metode for ansvarsdrevet OO

- **Inf1050 metoden (Iterativ):**
  - **Analyse av krav**
    - **(1) Identifiser aktører og deres mål**
    - **(2) Lag et høynivå bruksmønsterdiagram**
    - **(3) Spesifiser hvert bruksmønster tekstlig med hovedflyt og alternativ flyt**
  - **Objektdesign**
    - **For hvert bruksmønster:**
      - **(4) Identifiser objekter og fordel ansvar mellom dem (CRC)**
      - **(5) Lag sekvensdiagram for hovedflyt og viktige variasjoner**
      - **(6) Lag klassediagram som tilsvarer sekvensdiagrammene**
    - **(7) Lag til slutt klassediagram på systemnivå**



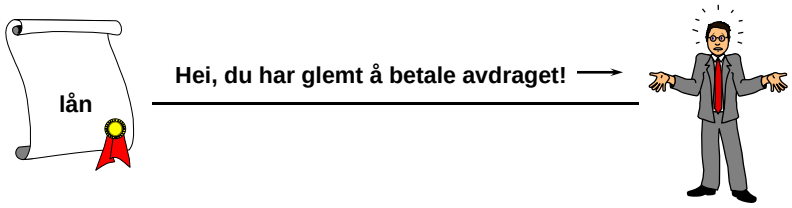
# Hva er et objekt

- **Et objekt er en representasjon av en virkelig “ting”.**
- **Et objekt har en entydig identitet, en indre tilstand, og evnen til å reagere på meldinger utenfra.**
- **Et objekt har altså “liv”.**
- **I modeller er alt mulig - også å “bevisstgjøre” i utgangspunktet døde ting som innsjøer, veier, kommuner, firmaer, lån ...**

# Det bevisste lån

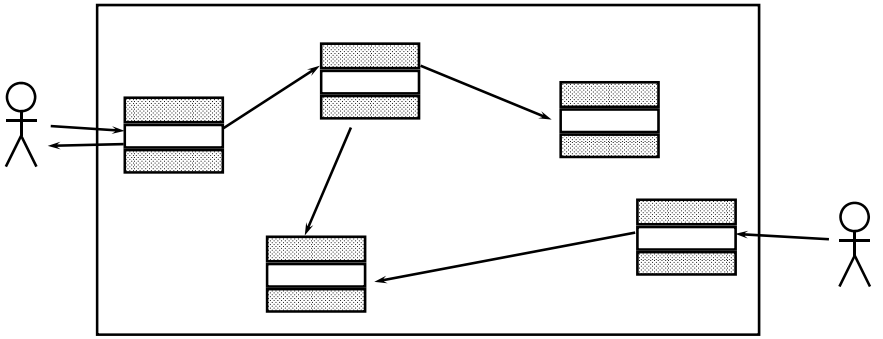
**Et låneobjekt kan f.eks.**

- kjenne til sin egen saldo, rentefot, forfallsdatoer osv.
- forrente seg selv
- purre på avdrag
- akseptere innbetalinger






# Utfordringen i å lage OO-modeller



**Gitt et sett bruksmønstre: Hvordan finne de "riktige" objektene og fordele ansvar mellom dem slik at bruksmønstrene blir realisert!**



# Hvordan finne objekter?

- Ta først utgangspunkt i "språket" til domenet (problemområdet):
  - F.eks. "produkt", "ingrediens", "kunde", "student", "konto", "innskudd", "reservasjon"
  - Finn begreper i bruksmønsterspesifikasjonene!
- Lag CRC-kort for objektene du tror du trenger
- Lag sekvensdiagrammer for bruksmønstrene



# Tre typer objekter

- Forretningsobjekter ("entity objects")
- Kontrollobjekter ("control objects")
- Kantobjekter ("boundary objects")
- Litt forenklet kan man si at denne tredelingen skiller mellom 1) objekter som skal *lagres* i en database, 2) objekter som *koordinerer* handlingene i et bruksmønster og 3) objekter som *kommuniserer* med aktørene.

## Forretningsobjekter ("entity objects")

- Representerer de "tingene" virksomheten håndterer, som for eksempel vare, tilbud, ordre, kunde osv.
- En forekomst av et forretningsobjekt kan leve lenge - kanskje like lenge som virksomheten!
- I motsetning til kantobjekter og kontrollobjekter lagres forretningsobjektene i en database (de er "persistente")



## Kontrollobjekter ("control objects")

- Representerer noe som gjøres i virksomheten
- Et kontrollobjekt lever vanligvis ikke lenger enn det handlingsforløpet det inngår i.
- Inf1050: ett kontrollobjekt pr. bruksmønster.
- Inf1050: Navnet på kontrollobjektet = navnet på bruksmønsteret!



## Kantobjekter (boundary objects)

- Kantobjekter aktiveres av handlinger fra aktører via brukergrensesnittet
  - for eksempel når aktøren ønsker å starte et bruksmønster ved å trykke på "Meld på kurs"-knappen i brukergrensesnittet.
- Kantobjektet oppretter deretter en forekomst av kontrollobjektet som kontrollerer selve handlingsforløpet i bruksmønsteret
- Vi kan godt tenke på kantobjekter som bindeleddet mellom et uspesifisert brukergrensesnitt og et kontrollobjekt. Kantobjekter kommuniserer kun med aktører (via brukergrensesnittet) og kontrollobjekter.



## CRC-kort (Class-Responsibility-Collaboration)

| Navn på objektet<br><<type objekt>> | Liste over samarbeidende objekter<br>(objekter som jeg utveksler meldinger med) |
|---|---|
| Ansvar:<br>• Hva objektet må vite<br>• Hva objektet skal gjøre | |

## Eks: CRC-kort for Student

| Student<br><<entity>> | Kurs |
|---|---|
| Vet min studentID<br>Vet hvilke kurs jeg har bestått<br>Vet hvilke kurs jeg er meldt opp til<br>Vet hvilke kurs jeg står på venteliste på<br>*+ andre ansvarsområder: Vil bli tydeligere etter hvert som man "designer" bruksmønstre ved hjelp av sekvensdiagrammer* | |



## Eks: CRC-kort for Universitet
## Objektorienterte systemer inneholder ofte et slikt "oppslagsobjekt"

| Universitet<br><<entity>> | Emne<br>Student |
|---|---|
| Oppslagsobjekt som er tilgjengelig for alle andre objekter:<br>Vet hvilke Emner og Studenter som finnes på universitetet<br>Har metode for å finne en gitt student<br>Har metode for å finne et gitt emne | |



## Eks: CRC-kort for bruksmønsteret "Meld på kurs"

| Kant<br><<boundary>> | MeldPaaKurs |
|---|---|
| Kommuniserer med aktøren og kontrollobjektet | |

| Universitet<br><<entity>> | Emne<br>Student |
|---|---|
| Oppslagsobjekt som vet hvilke emner og studenter som finnes i systemet | |

| Emne<br><<entity>> | Kurs |
|---|---|
| Vet min emnekode<br>Vet hvilke emner som forutsettes<br>Vet hvilke kurs som holdes i emnet | |

| MeldPaaKurs<br><<control>> | Universitet<br>Emne<br>Kurs<br>Student<br>Kant |
|---|---|
| Kontrollerer hendelses-forløpet i bruksmønsteret "Meld på kurs" | |

| Student<br><<entity>> | Kurs |
|---|---|
| Vet min studentID<br>Vet hvilke kurs jeg har bestått<br>Vet hvilke kurs jeg er meldt opp til<br>Vet hvilke kurs jeg står på venteliste på | |

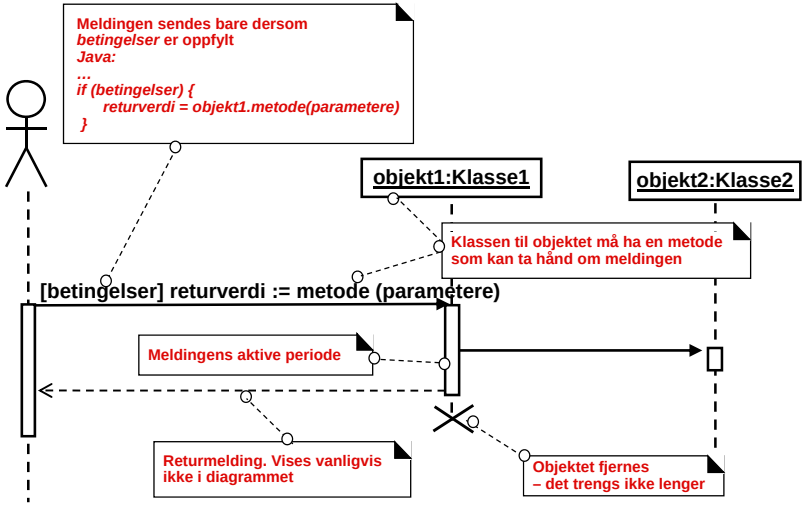
| Kurs<br><<entity>> | Emne<br>Student |
|---|---|
| Vet semesteret som (et kurs i) et bestemt emne holdes<br>Vet antall plasser<br>Vet hvilke studenter som er påmeldt<br>Vet hvilke studenter som står på venteliste | |



# Objektdesign med UML sekvensdiagrammer

- Et UML sekvensdiagram viser en interaksjon mellom aktører og objekter i systemet for et bestemt bruksmønster
  - Fokuserer på hvordan objektene samarbeider for å løse en bestemt oppgave (bruksmønster)
  - Er ofte nyttig for å identifisere (og spesifisere bruken av) metodene til objektene i systemet

# Syntaks for UML sekvensdiagram





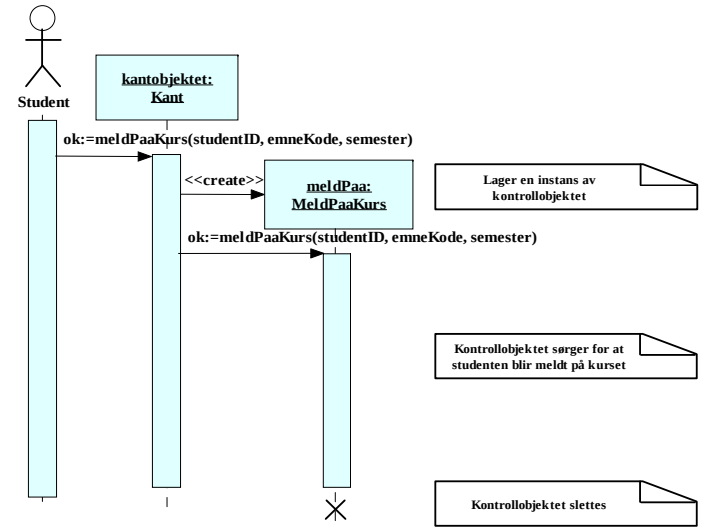

# Sammenheng mellom bruksmønster og sekvensdiagram

- For hvert bruksmønster lager du (i de aller fleste tilfeller) et sekvensdiagram for hovedflyt
- For hver alternative flyt kan du velge å lage et nytt sekvensdiagram.
  - Det er viktig å lage sekvensdiagram for variasjoner som har stor innvirkning på designet
    - Introduserer variasjonen nye objekter?
    - Introduserer variasjonen nye metoder?



# Hovedflyt for "Meld på kurs"

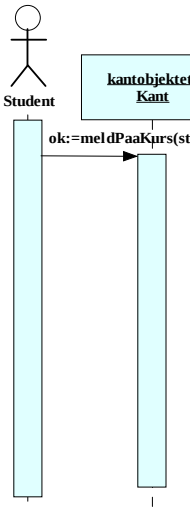
(iterasjon #1, uten forretningsobjektene)






(infostoff)

## Eksempel i Java: metoden *meldPaaKurs* til kantobjektet

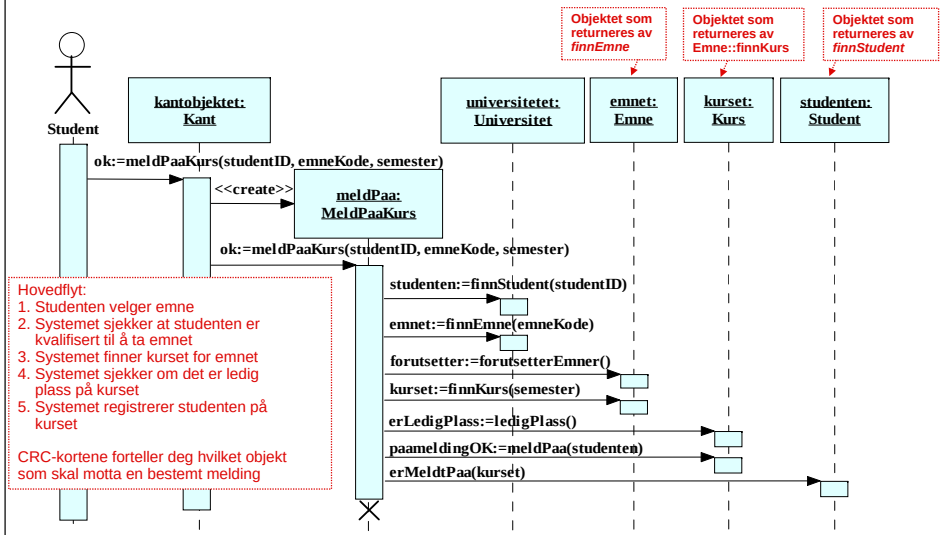



```
boolean meldPaaKurs(String studentID, String emneKode, String semester)
{
    MeldPaaKurs meldPaa;
    boolean ok;

    // kantobjektet lager et nytt kontrollobjekt:
    meldPaa = new MeldPaaKurs(this);

    // så overtar kontrollobjektet handlingsforløpet i bruksmønsteret:
    ok = meldPaa.meldPaaKurs(studentID, emneKode, semester);
    return ok;
}
```

# Hovedflyt for ”Meld på kurs”





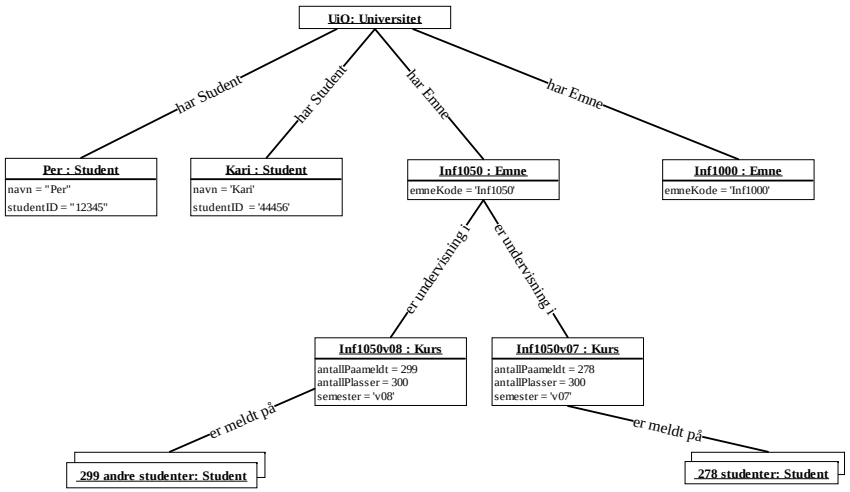

(infostoff - NB! Ikke komplett kode)

# Eksempel i Java: metoden meldPaaKurs til kontrollobjektet

```
public class MeldPaaKurs {
  ...
  public boolean meldPaaKurs(String studentID, String emneKode, String semester)
  {
    // antar at objektet ”universitetet” er globalt tilgjengelig:
    Student studenten = universitetet.finnStudent(studentID);
    Emne emnet = universitetet.finnEmne(emneKode);
    boolean forutsetteremner = emnet.forutsetterEmner();
    Kurs kurset = emnet.finnKurs(semester);
    boolean erLedigPlass = kurset.ledigPlass();
    boolean paameldingOK = kurset.meldPaa(studenten);
    studenten.erMeldtPaa(kurset);
    return paameldingOK;
  }
}
```

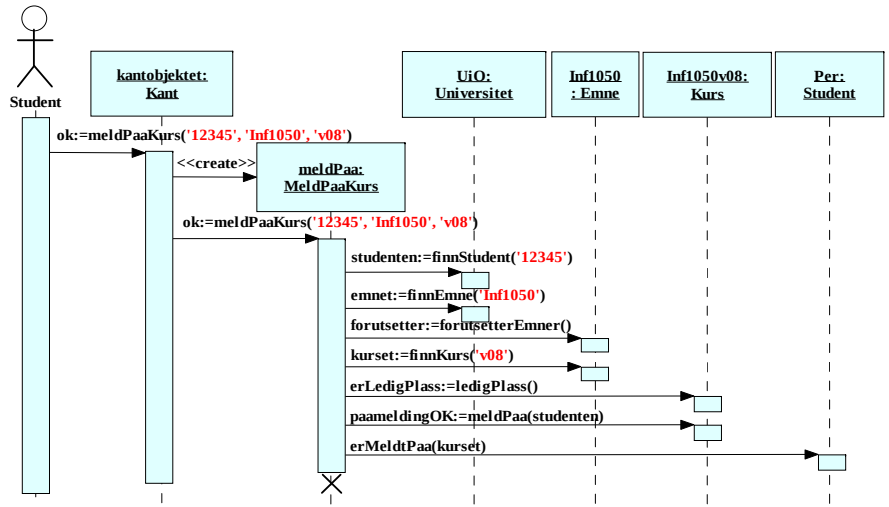


# Objektdiagram, før Per melder seg på Inf1050

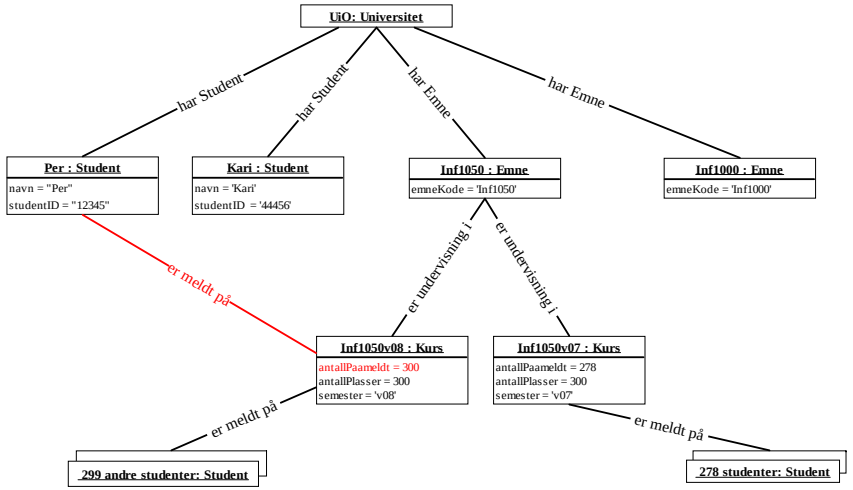





# … og så registrerer Per seg på Inf1050 v08



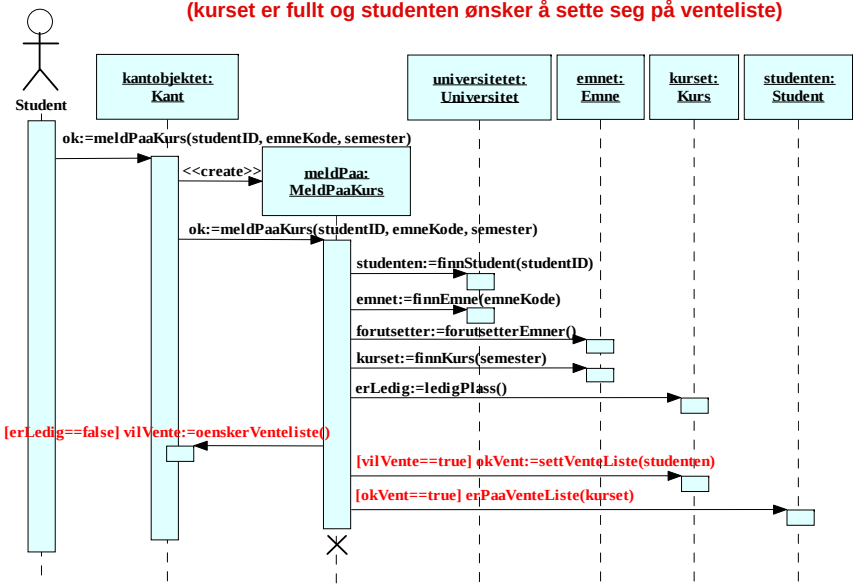
# Objektdiagram, etter at Per har meldt seg på Inf1050 v08






# Alternativ flyt, steg 4

(kurset er fullt og studenten ønsker å sette seg på venteliste)






(infostoff - NB! Ikke komplett kode)

# Java tilsvarende steg 1-3 + alternativ flyt, steg 4

(kurset er fullt og studenten ønsker å sette seg på venteliste)

```
public class MeldPaaKurs {
    private Kant kantobjektet;
    public boolean meldPaaKurs(String studentID, String emneKode, String semester) {
        Student studenten = universitetet.finnStudent(studentID);
        Emne emnet = universitetet.finnEmne(emneKode);
        boolean forutsetteremner = emnet.forutsetterEmner();
        Kurs kurset = emnet.finnKurs(semester);
        boolean erLedigPlass = kurset.ledigPlass();
        if (erLedigPlass == false) {
            boolean vilVente = kantobjektet.oenskerVenteliste();
            if (vilVente == true) {
                boolean okVent= kurset.settVenteListe(studenten);
                if (okVent == true) {
                    studenten.erPaaVenteliste(kurset);
                    return true;
                }
            }
        }
    }
}
```



# Objektdiagram, etter at Kari ble satt på venteliste