Fra krav til objektdesign

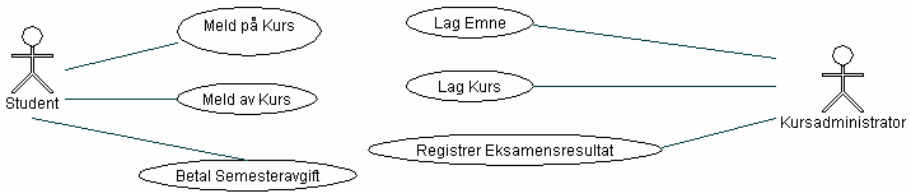
Ansvarsdrevet OO:

CRC og UML Sekvensdiagrammer

Dagens forelesning

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

Kursregistrering bruksmønstermodell (ny versjon)



Endringer i modellen sammenlignet med tidligere forslag:
1) Vi skiller mellom "Emne" og "Kurs"
2) "Lag Emne" og "Lag Kurs" inneholder "Nytt Emne" og "Nytt Kurs" som variasjoner
3) Bruksmønster "Behandle dispensasjonssøknad" utgår
4) Variasjon på "Meld på kurs":
Dersom et kurs er fullt kan studenten velge å bli satt på venteliste.
5) Variasjon på "Meld av kurs":
Det finnes studenter på venteliste: Første på ventelista får plassen

Spesifikasjon av "Lag emne"

Navn: Lag emne

Aktør: Kursadministrator

Trigger: Kursadministrator ønsker å opprette eller endre et emne

Normal Hendelsesflyt:

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

…

Variasjoner:

2a. Emnekoden eksisterer ikke:

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.

Revidert spesifikasjon av ”Meld på kurs”

Navn: Meld på kurs

Aktør: Student

Trigger: Student ønsker å melde seg på et kurs

Pre-betingelse: Student har betalt semesteravgift og er logget inn på systemet

Post-betingelse: Student er meldt på kurset eller er satt på venteliste

Normal Hendelsesflyt:

1. Studenten velger emne
2. Systemet sjekker at studenten kvalifiserer 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.)

Variasjoner:

1a. Emnet finnes ikke:

1. Studenten velger et annet emne eller avslutter

2a. Emnet forutsetter andre emner:

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

1a. Studenten har ikke bestått kurs for emner som forutsettes:

1. Studenten velger et annet emne eller avslutter

3a. Det holdes ikke kurs i emnet dette semesteret:

1. Studenten velger et annet emne eller avslutter

4a. Kurset er fullt:

1. Systemet spør om studenten ønsker å bli satt på venteliste

1a. Studenten ønsker å bli satt på venteliste:

1. Systemet setter studenten på venteliste

Relatert informasjon:

I denne versjonen holdes administrasjon av gruppeundervisning utenfor systemet

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 normal hendelsesflyt og variasjoner**
  - **Objektdesign**
    - **For hvert bruksmønster:**
      - **(4) Identifiser objekter og fordel ansvar mellom dem (CRC)**
      - **(5) Lag sekvensdiagram for normal hendelsesflyt og viktige variasjoner**
      - **(6) Lag klassediagram som tilsvarer sekvensdiagrammene**
    - **(7) Lag til slutt klassediagram på systemnivå**

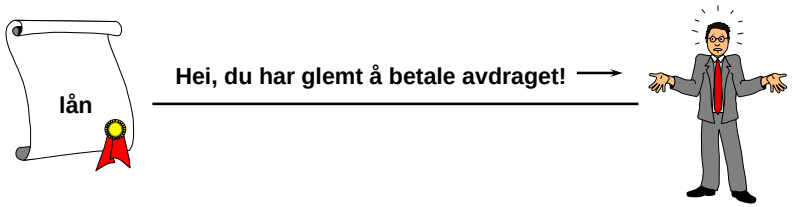
Hva er et objekt

- **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”. Virkelighetens objekter er dyr, planter, maskiner og mekanismer.**
- **I modeller er alt mulig - også å “bevisstgjøre” i utgangspunktet døde “ting” som innsjøer, veier, kommuner, firmaer, lån ...**

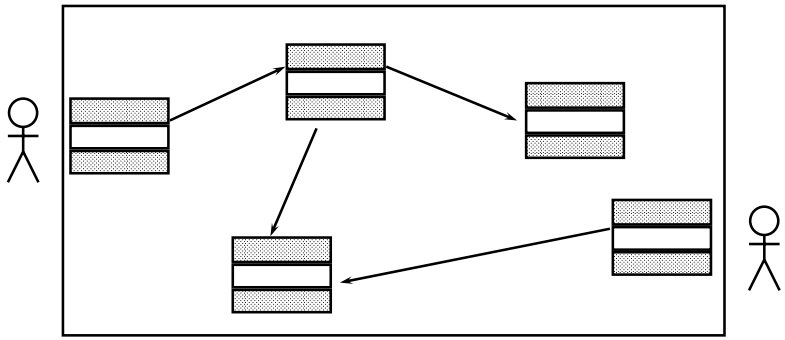
Det bevisste lån

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 objekter 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”
  - 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 <<type objekt>>	Liste over samarbeidende objekter (objekter som kan delegeres oppgaver)
Ansvar: • Hva objektet må vite • Hva objektet skal gjøre	

Eks: CRC-kort for Student

Student <<entity>>	Kurs
Vet min studentID Vet hvilke kurs jeg har tatt Vet hvilke kurs jeg er meldt opp til Vet hvilke kurs jeg står på venteliste på *+ 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 <<entity>>	Emne Student
Oppslagsobjekt som er tilgjengelig for alle andre objekter: Vet hvilke Emner og Studenter som finnes på universitetet Har metode for å finne en gitt student Har metode for å finne et gitt emne	

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

Kant <<boundary>>	MeldPaaKurs
Kommuniserer med aktøren og kontrollobjektet	

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

Emne <<entity>>	Kurs
Vet min emnekode Vet hvilke emner som forutsettes Vet hvordan man finner kurs i emnet	

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

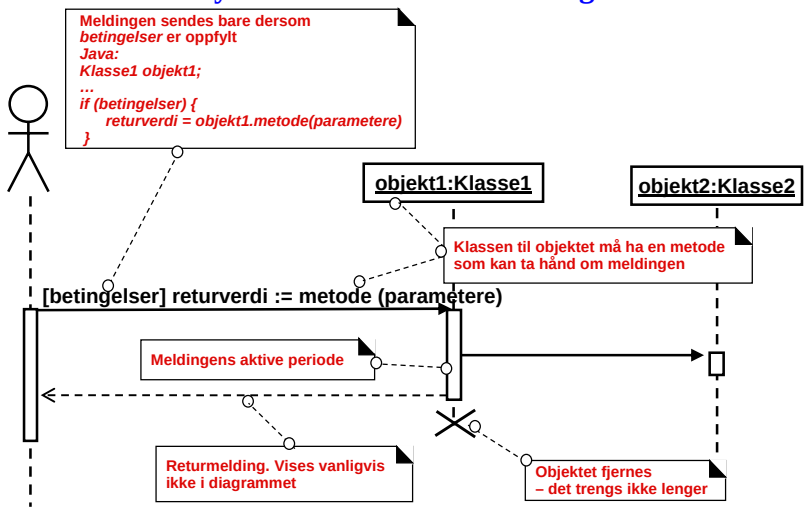
Student <<entity>>	Kurs
Vet min studentID Vet hvilke kurs jeg har tatt Vet hvilke kurs jeg er meldt opp til Vet hvilke kurs jeg står på venteliste på	

Kurs <<entity>>	Emne Student
Vet semesteret som (et kurs i) et bestemt emne holdes Vet antall plasser Vet hvilke studenter som er påmeldt 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

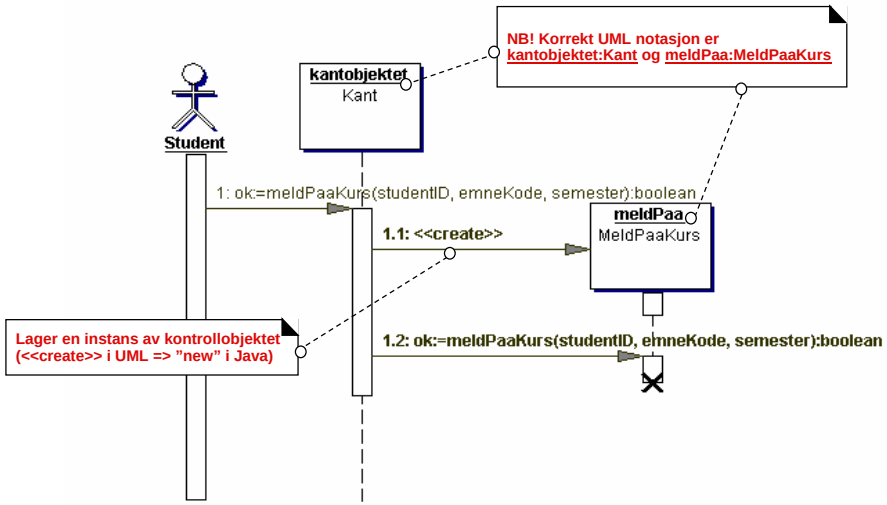


Sammenheng mellom bruksmønster og sekvensdiagram

- For hvert bruksmønster lager du (i de aller fleste tilfeller) et sekvensdiagram for normal hendelsesflyt
- For hver variasjon 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?

Normal hendelsesflyt for "Meld på kurs"

(iterasjon #1, uten forretningsobjektene)

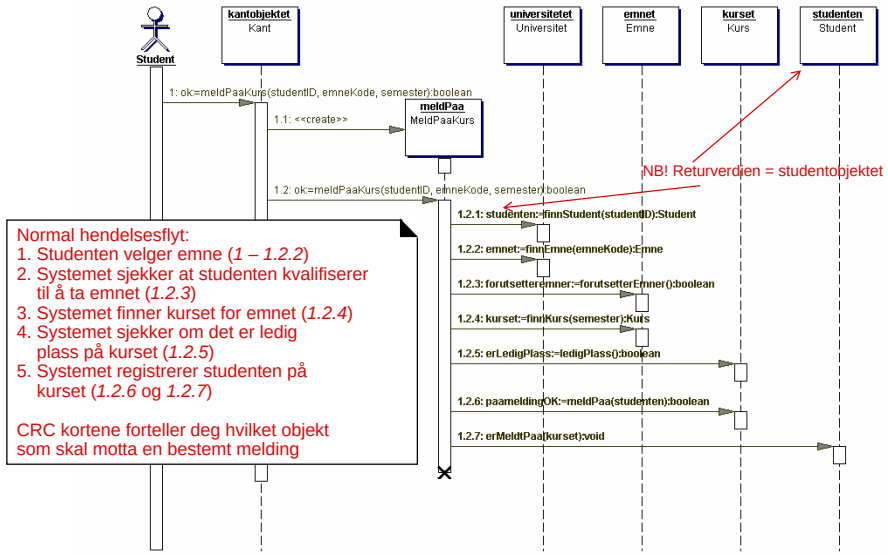


Normal hendelsesflyt for "Meld på kurs"

(Eksempel på Java kode for metoden meldPaaKurs til kantobjektet)

```
public class Kant {

   ...

   boolean meldPaaKurs(String studentID, String emneKode, String semester) {

      // message #1.1 (det kan tenkes at kontrollobjektet må sende meldinger til
      // kantobjektet.  Derfor sendes kantobjektet (this) inn som parameter til
      // kontrollobjektets  constructor):

      MeldPaaKurs meldPaa = new MeldPaaKurs(this);

      // message #1.2 (her utføres selve logikken i bruksmønsteret):

      boolean ok = meldPaa.meldPaaKurs(studentID, emneKode, semester);

      return ok;

   }

}
```

Normal hendelsesflyt for ”Meld på kurs”



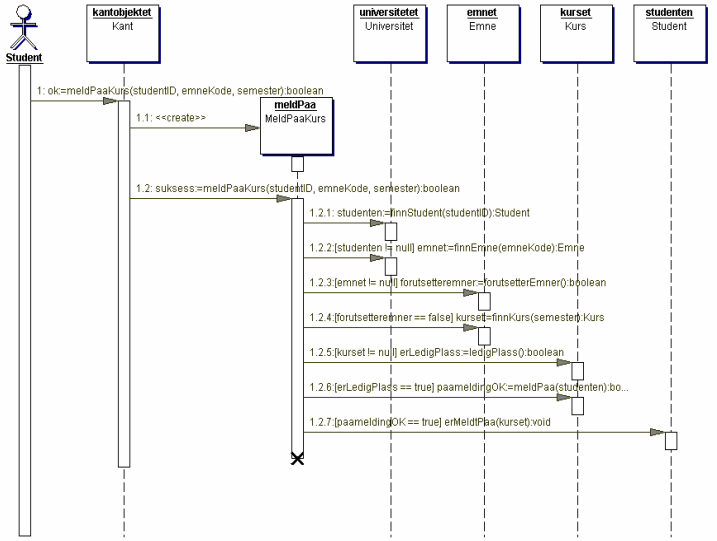
Normal hendelsesflyt for ”Meld på kurs”

(Eksempel på Java kode for metoden meldPaaKurs til kontrollobjektet)

```
public class MeldPaaKurs {
  …
  public boolean meldPaaKurs(String studentID, String emneKode, String semester)
  {
    // antar at objektet ”universitetet” er tilgjengelig:
    Student studenten = universitetet.finnStudent(studentID);     // message #1.2.1
    Emne emnet = universitetet.finnEmne(emneKode);                // message #1.2.2
    boolean forutsetteremner = emnet.forutsetterEmner();          // message #1.2.3
    Kurs kurset = emnet.finnKurs(semester);                       // message #1.2.4
    boolean erLedigPlass = kurset.ledigPlass();                   // message #1.2.5
    boolean paameldingOK = kurset.meldPaa(studenten);             // message #1.2.6
    studenten.erMeldtPaa(kurset);                                 // message #1.2.7
  }
}
```


Normal hendelsesflyt for ”Meld på kurs”

(ytterligere detaljering: sekvensdiagram med betingelser)



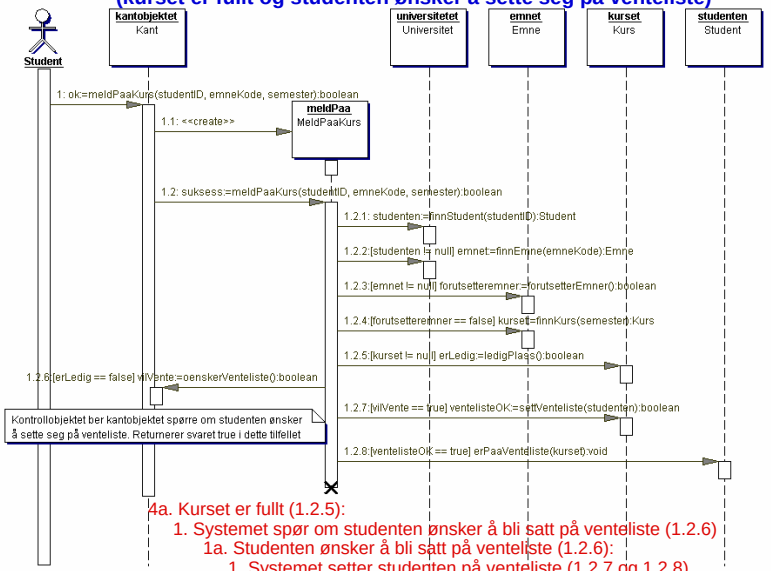
Normal hendelsesflyt for ”Meld på kurs”

(Eksempel på Java kode for metoden meldPaaKurs fra forrige sekvensdiagram)

```
public class MeldPaaKurs {
  // i Inf1050 bruker vi konvensjonen at betingelsene også gjelder de resterende meldinger i sekvensdiagrammet, dvs:
  public boolean meldPaaKurs(String studentID, String emneKode, String semester) {
    Student studenten = universitetet.finnStudent(studentID);          // message #1.2.1
    if (studenten != null) {
      Emne emnet = universitetet.finnEmne (emneKode);                  // message #1.2.2
      if (emnet != null {
        boolean forutsetteremner = emnet.forutsetterEmner();           // message #1.2.3
        if (forutsetteremner == false) {
          Kurs kurset = emnet.finnKurs(semester );                     // message #1.2.4
          if (kurset != null) {
            boolean erLedigPlass = kurset.ledigPlass();                // message #1.2.5
            if (erLedigPlass == true) {
              boolean paameldingOK = kurset.meldPaa(studenten);        // message #1.2.6
              if (paameldingOK == true) {
                studenten.erMeldtPaa(kurset);                          // message #1.2.7
    …} else { <de forskjellige variasjonene som sekvensdiagrammet IKKE definerer>} }}}}}
}
```

Variasjon 4a.1.1a.1

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



4a. Kurset er fullt (1.2.5):
1. Systemet spør om studenten ønsker å bli satt på venteliste (1.2.6)
1a. Studenten ønsker å bli satt på venteliste (1.2.6):
1. Systemet setter studenten på venteliste (1.2.7 og 1.2.8)

Variasjon 4a.1.1a.1

(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) {

        // SAMME KODE SOM PÅ "FOIL 28"  FRAM TIL MESSAGE 1.2.5, OG DERETTER:

        boolean erLedigPlass = kurset.ledigPlass();                    // message #1.2.5

        if (erLedigPlass == false) {

            boolean vilVente = kantobjektet..oenskerVenteliste();      // message #1.2.6

            if (vilVente == true) {

                boolean venteListeOK = kurset.settVenteListe(studenten);   // message #1.2.7

                if (venteListeOK == true) {

                    studenten.erPaaVenteliste(kurset);                 // message #1.2.8

                }

            }

        }

    }

}
```