Dagens forelesning

- Litt mer om design med UML sekvensdiagrammer
  - Sentralisert og delegert kontrollstil
    - Resultater fra et eksperiment
- UML klassediagrammer
  - Notasjon: UML klassediagram og objektdiagram
  - Metode: Fra sekvensdiagram til klassediagram

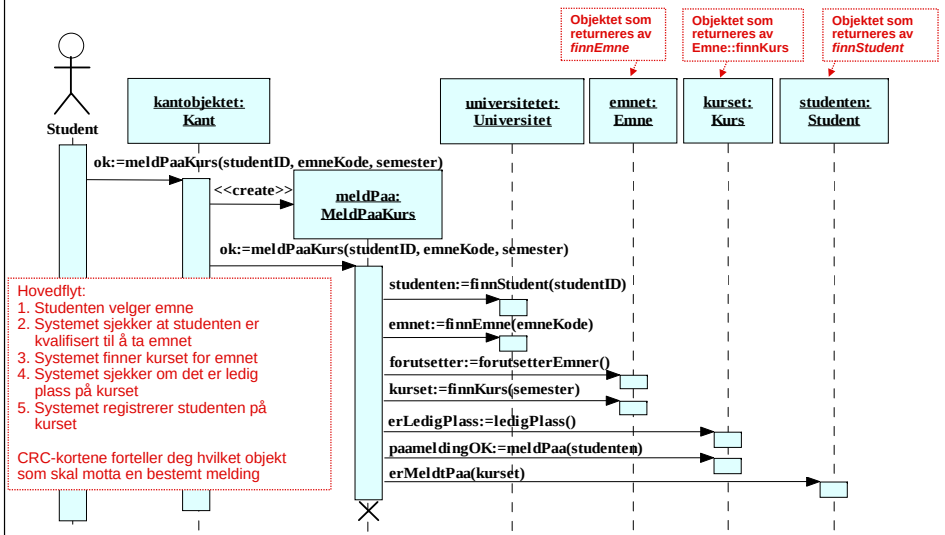
Metode for ansvarsdrevet OO med UML

- 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

Delegering av ansvar i en trelagsarkitektur

- Forretningsobjekter (“entity objects”)
- Kontrollobjekter (”control objects”)
- Kantobjekter (“boundary objects”)
- Hvor mye ansvar bør kontrollobjektene ha, og i hvilken grad bør vi ”bevisstgjøre” forretningsobjektene våre??

Hovedflyt for ”Meld på kurs”



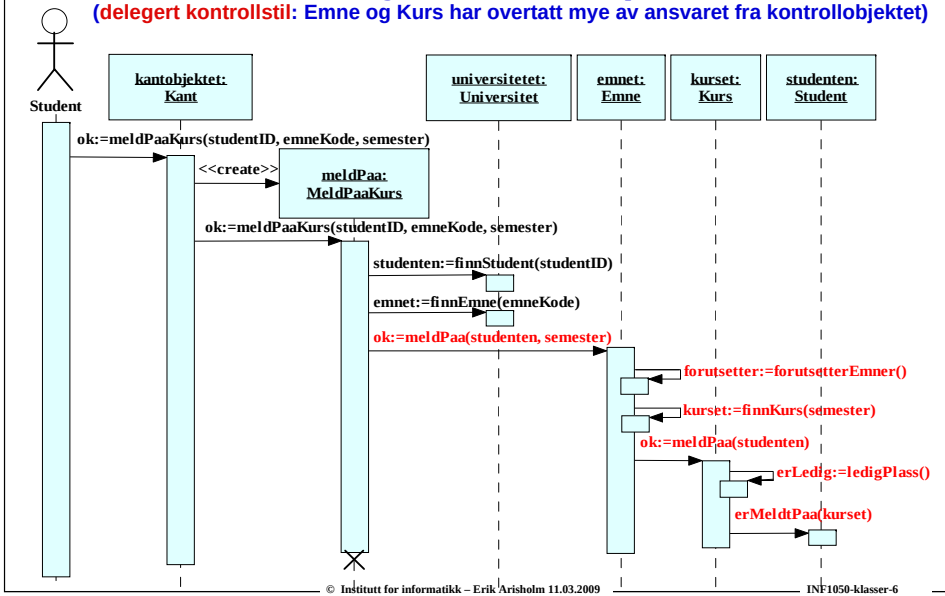
Hovedflyt for "Meld på kurs"

(Eksempel på Java kode for metoden meldPaaKurs i en sentralisert kontrollstil)

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


Hovedflyt for "Meld på kurs"

(delegert kontrollstil: Emne og Kurs har overtatt mye av ansvaret fra kontrollobjektet)

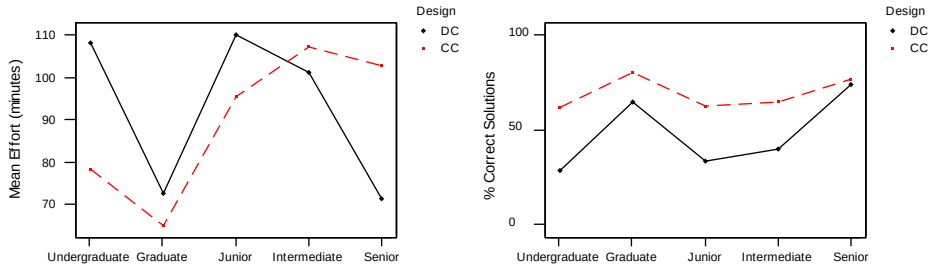


Hovedflyt for "Meld på kurs"

(Eksempel på Java kode for metoden meldPaaKurs i en delegert kontrollstil)

```
public class MeldPaaKurs {
  ...
  public boolean meldPaaKurs(String studentID, String emneKode, String semester)
  {
    // antar at objektet "universitetet" er tilgjengelig:
    Student studenten = universitetet.finnStudent(studentID);
    Emne emnet = universitetet.finnEmne(emneKode);
    boolean ok = emnet.meldPaa(studenten, semester);
    return ok;
  }
}
```


Resultater fra et kontrollert eksperiment (*)



DC = Delegated Control Style
CC = Centralized Control Style

Totalt 158 Java-utviklere deltok, og skulle gjøre endringer på enten et *DC* eller et *CC* design alternativ for det samme systemet.

Vi målte tid ("Mean Effort") og kvalitet ("% correct solutions")

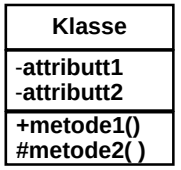
Kun seniorkonsulentene hadde klare fordeler av et delegert (DC) design

*** Erik Arisholm and Dag Sjøberg, "Evaluating the Effect of a Delegated versus Centralized Control Style on the Maintainability of Object-Oriented Software," *IEEE Transactions on Software Engineering*, 2004**

Delegert vs sentralisert kontrollstil

- **Sentralisert kontrollstil:**
  - Lett å få oversikt over hva som skjer i et bruksmønster
  - Feilsituasjoner/variasjoner som krever tilbakemeldinger fra en aktør (via kantobjektene) kan enkelt håndteres av kontrollobjektet
  - Introduserer flere avhengigheter mellom kontrollobjekt og forretningsobjekter. Potensielt mindre gjenbrukbar/vedlikeholdbar kode
- **Delegert kontrollstil:**
  - Mer ”elegant” objektorientert design, men:
  - Overdreven bruk av delegering gjør det vanskelig å få oversikt (spesielt dersom sekvensdiagrammer ikke er tilgjengelige!)
  - Litt mer komplisert å håndtere feilsituasjoner/variasjoner som krever tilbakemeldinger fra en aktør (siden all kommunikasjon med kantobjektene må gå via kontrollobjektene)

UML – Klasser og objekter



+ betyr ”public”

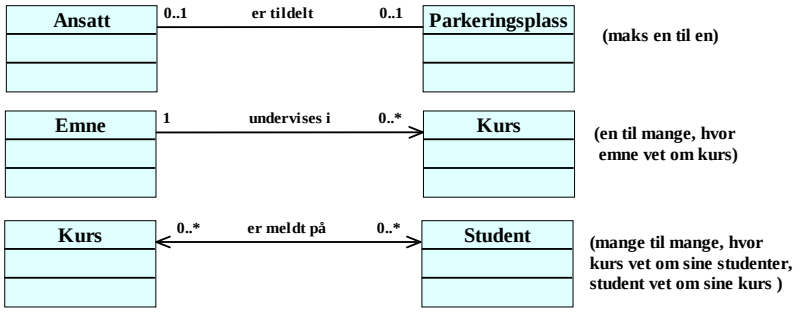
- betyr ”private”

betyr ”protected”

En klasse beskriver hva objektene vet (tilstand/attributter) og hvilke meldinger de forstår (metoder).

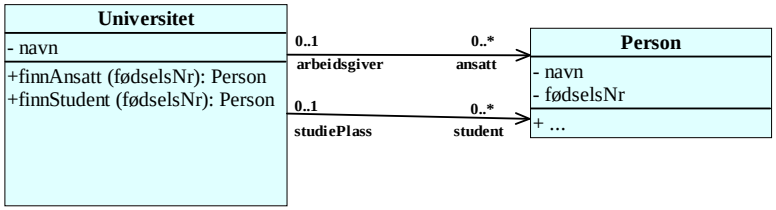
Objektene er forekomster av klassebeskrivelsen.

Assosiasjoner mellom to klasser



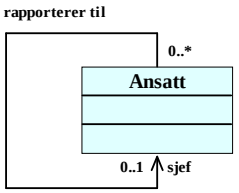
Indikator	Antall forekomster
0..1	Null eller 1
1	nøyaktig 1
n	nøyaktig n
0..*	0 eller flere
*	0 eller flere
1..*	1 eller flere
1..n	1 eller flere (<= n)

Roller



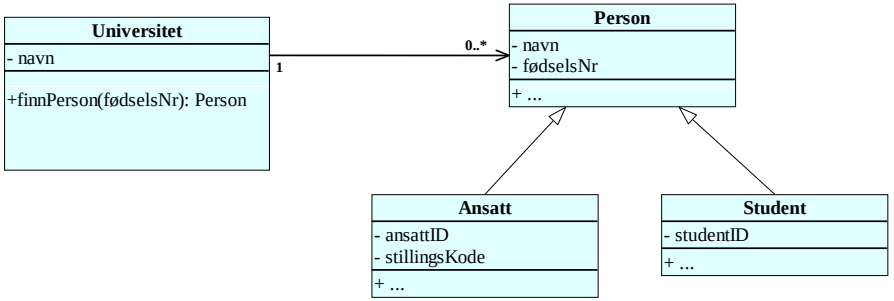
En person kan ha rollen som *ansatt* på maks ett universitet
En person kan ha rollen som *student* på maks ett universitet

(men Per kan godt være student på UiO og ansatt ved NTNU)

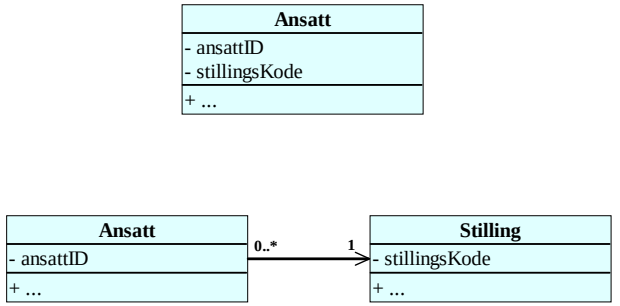


En ansatt rapporterer til maks en annen ansatt, som er sjef

Arv

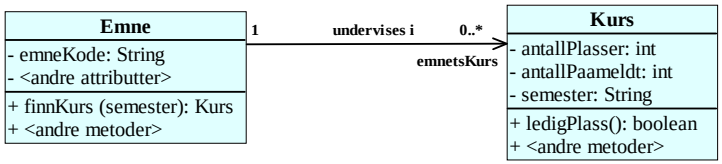


Attributter eller klasser?

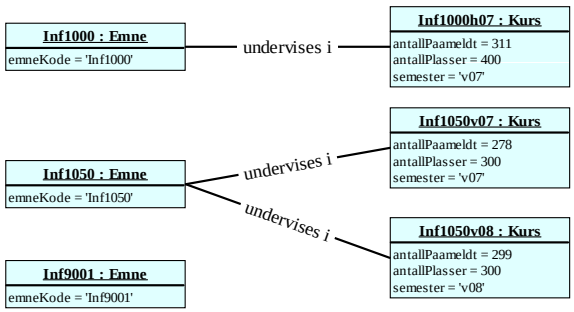


Ved å utvide stillingsKode-attributtet til å bli en egen klasse (Stilling) kan man definere andre attributter og metoder som har spesielt med stillinger å gjøre

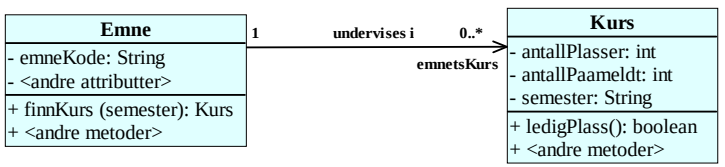
Klassediagram vs objektdiagram

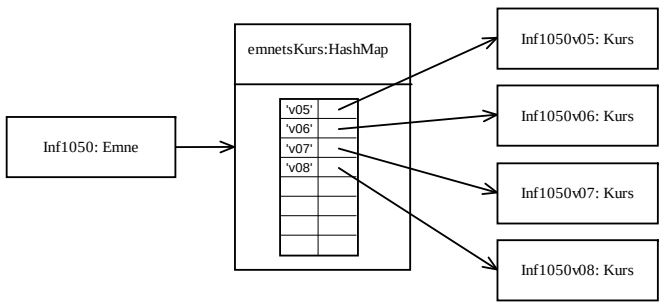


Multiplisitetene i klassediagrammet forteller oss at vi kan ha emner uten kurs, men ikke kurs uten emne

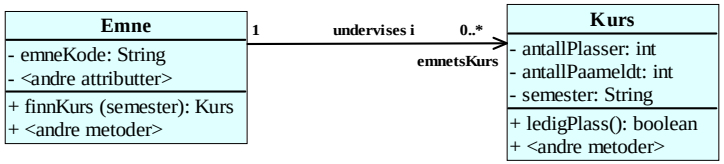


Infostoff: Eks. på realisering i Java





Infostoff: Eks. på realisering i Java

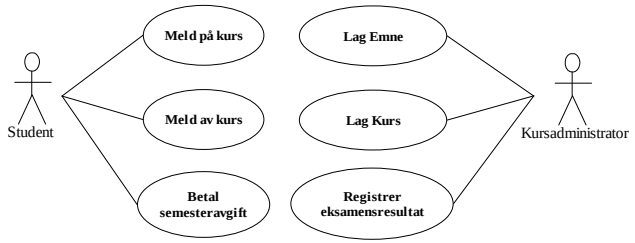



```
public class Emne {
    private String emneKode;
    <andre attributter>;
    private HashMap emnetsKurs = new HashMap();
    Kurs finnKurs(String semester) {
        Kurs kurset;
        kurset = (Kurs) emnetsKurs.get(semester);
        return kurset;
    }
    <andre metoder>
}
```


Sammenhengen mellom sekvensdiagram og klassediagram

- **Start med sekvensdiagrammet for hovedflyt:**
  - **Lag klasser for alle objektene i sekvensdiagrammet.**
  - **For hver melding til et bestemt objekt i sekvensdiagrammet: Legg til en tilsvarende metode for klassen til dette objektet.**
  - **Legg til de attributtene som hver av disse metodene trenger**
  - **Lag nødvendige assosiasjoner for at klassene skal kunne utføre sine metoder (f.eks. sende meldinger til andre objekter)**
- **For hvert sekvensdiagram for en variasjon:**
  - **Utvid klassediagrammet med nye klasser, metoder, attributter og assosiasjoner i klassediagrammet basert på meldingene i sekvensdiagrammet for variasjonen.**

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

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

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	

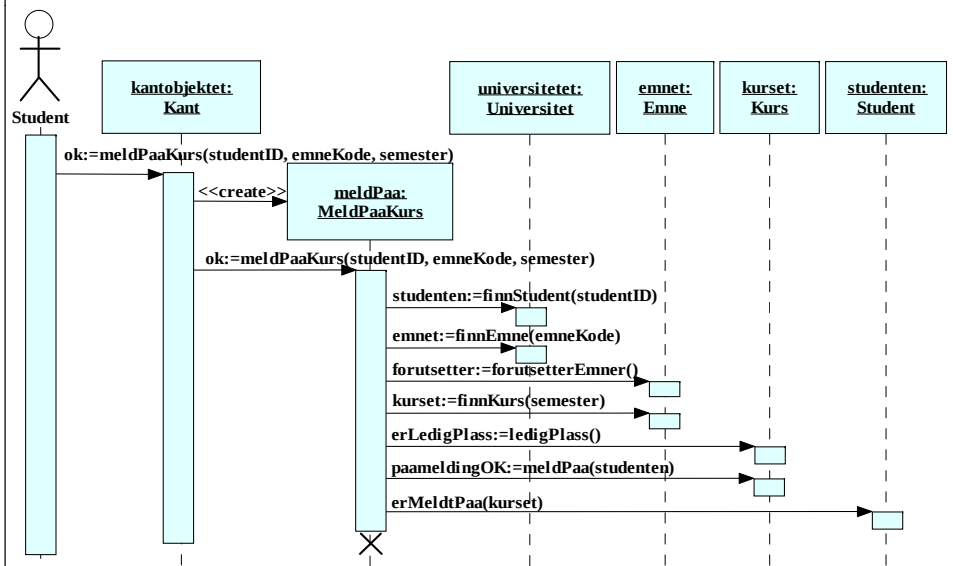
Emne <<entity>>	Kurs
Vet min emnekode Vet hvilke emner som forutsettes Vet hvordan man finner kurs i emnet	

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

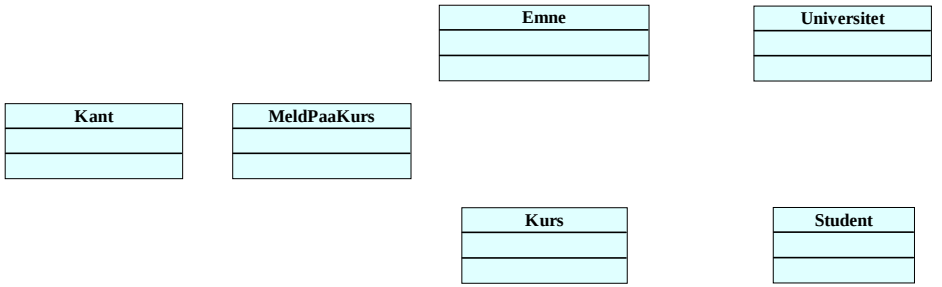
Student <<entity>>	Kurs
Vet min studentID Vet hvilke kurs jeg har tatt Vet hvilke kurs jeg er meldt opp til Vet hvilke kurs jeg står på venteliste på	

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	

Normal hendelsesflyt for ”Meld på kurs”

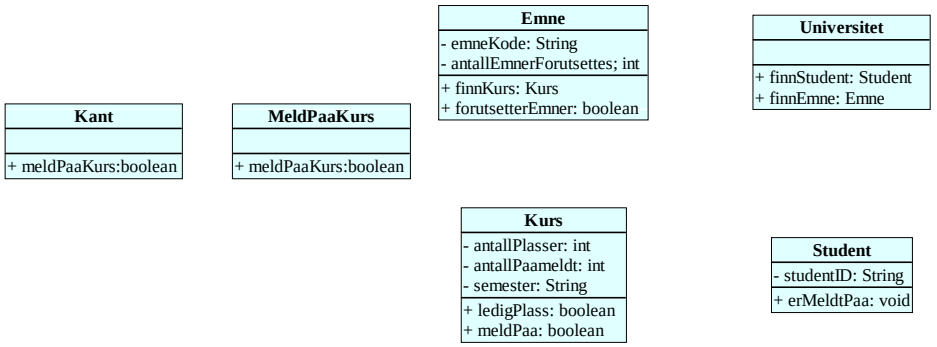


Klasser involvert i hovedflyt



1) Inkluder klassene du finner i sekvensdiagrammet

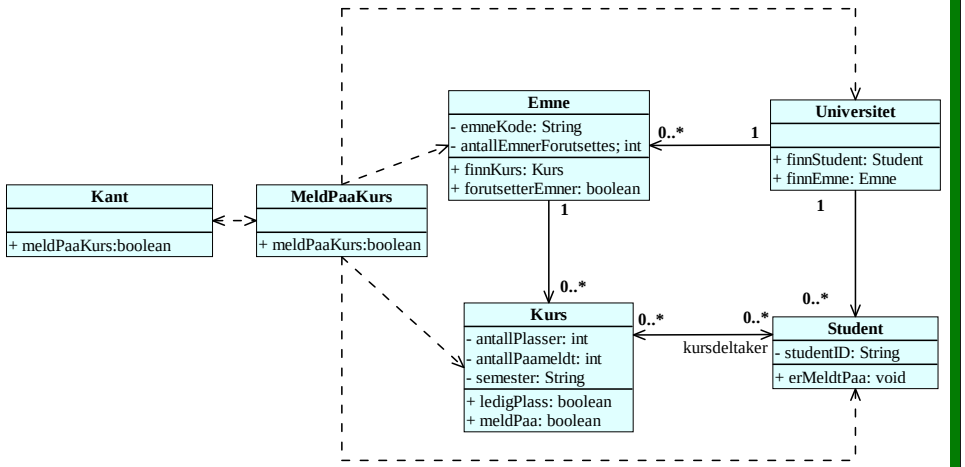
Metoder og attributter for hovedflyt



2) Inkluder metodene som ble brukt i sekvensdiagrammet

3) Inkluder attributter som metodene trenger

Komplett klassediagram for hovedflyt

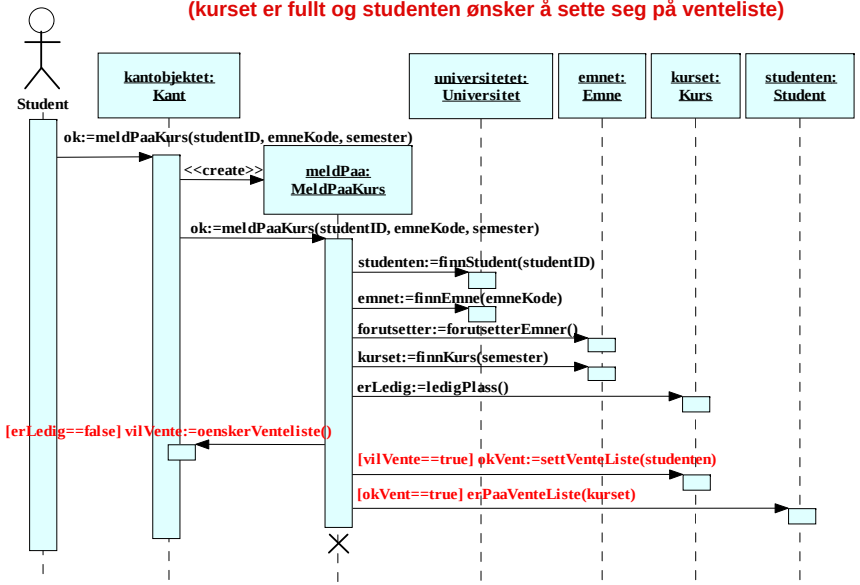


4) Inkluder assosiasjoner som metodene "trenger" (bruker eller oppretter)

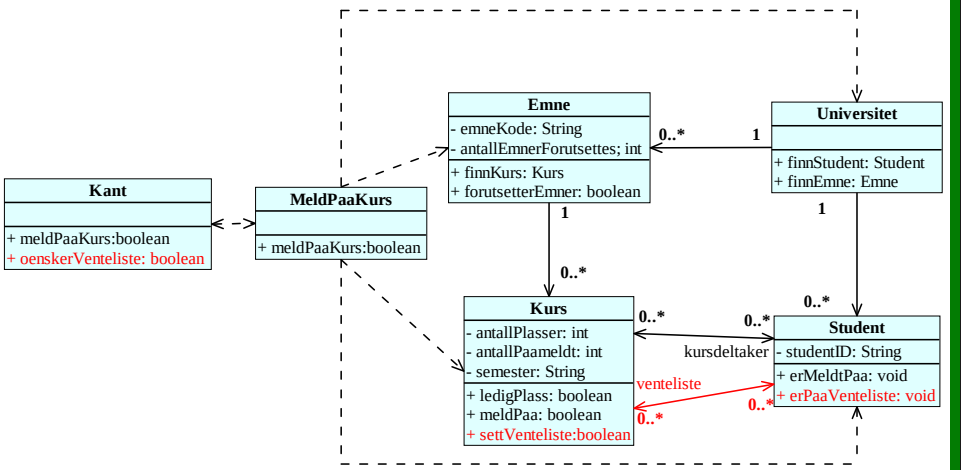
5) Inkluder avhengighetspiler mellom klassene som utveksler meldinger

Alternativ flyt, steg 4

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



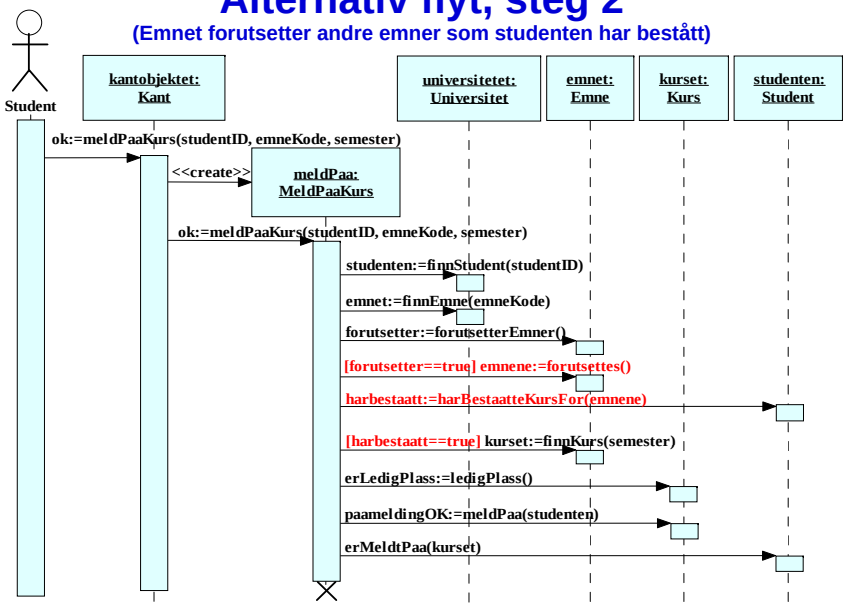
Klassediagram for hovedflyt og alternativ flyt (steg 4)



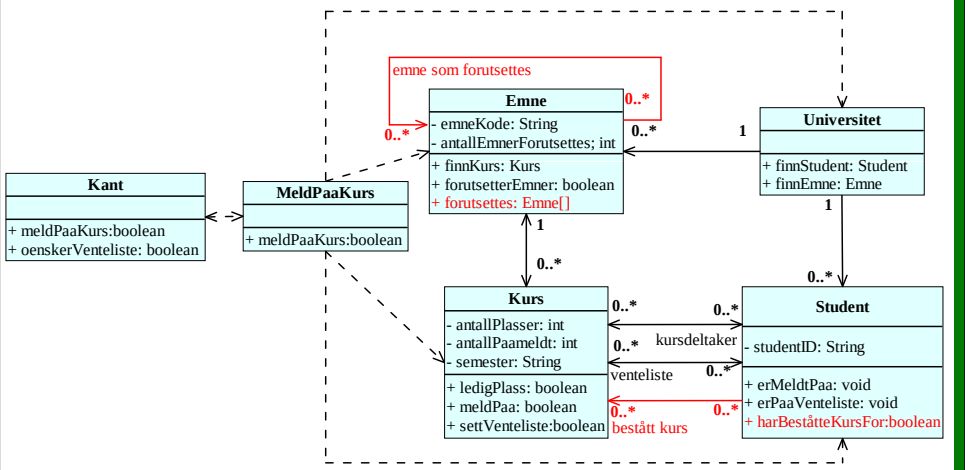
Legger til nye metoder og assosiasjoner for alternativ flyt

Alternativ flyt, steg 2

(Emnet forutsetter andre emner som studenten har bestått)

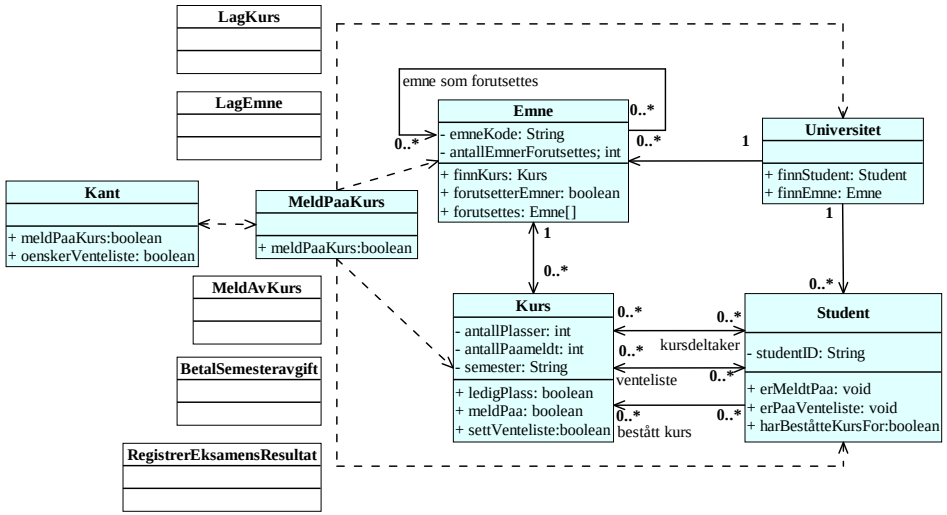


Klassediagram for normal hendelsesflyt + alternativ flyt steg 2 og 4



Legger til nye metoder og assosiasjoner for alternativ flyt steg 2

Klassediagram på systemnivå



De andre bruksmønstrene vil introdusere nye kontrollobjekter og evt. nye forretningsobjekter, samt nye assosiasjoner, metoder og attributter på eksisterende forretningsobjekter. Kantobjektet vil få flere metoder (antar ett kantobjekt i systemet)