Persistens

Erik Arisholm

Samling av trådene

Systemutvikling som helhet

1. Systemutvikling: motivasjon ..... Jo Hannay, Simula & Ifi
2. Systemutviklingsprosessen ..... Rune Steinberg, Visma Software AS
3. Prosjektledelse og prosjektarbeid ... Rune Steinberg, Visma Software AS

Kunde/leverandør/bruker-forhold

4. Kravhåndtering ..... Erik Arisholm, Simula & Ifi
5. Avtaler & kontrakter ... Jørgen Petersen, Promis AS
6. Estimering ..... Stein Grimstad, Simula
7. Jus & etikk .... Dag W. Schartum, Senter for Rettsinformatikk

Systemets struktur og design

8. Modellering av krav med use cases ...Erik Arisholm, Simula & Ifi
9. Objektorientert analyse (2 forel.) ..... Erik Arisholm, Simula & Ifi
10. Persistens og databaser .....Erik Arisholm, Simula & Ifi
11. Arkitektur ..... Dag Lorås, Visma Software AS

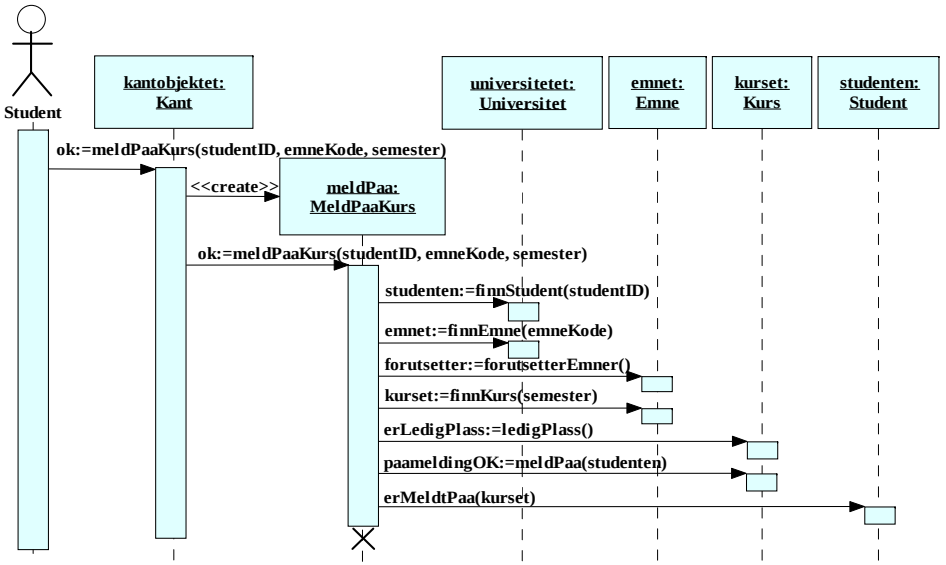
Koding, validering og vedlikehold

12. Modellbasert utvikling med Genova ... Esito AS
13. Validering og verifisering (2 forel.) ..... Lionel Briand, Simula & Ifi
14. Konfigurasjonsstyring..... Hans Christian Benestad, Simula

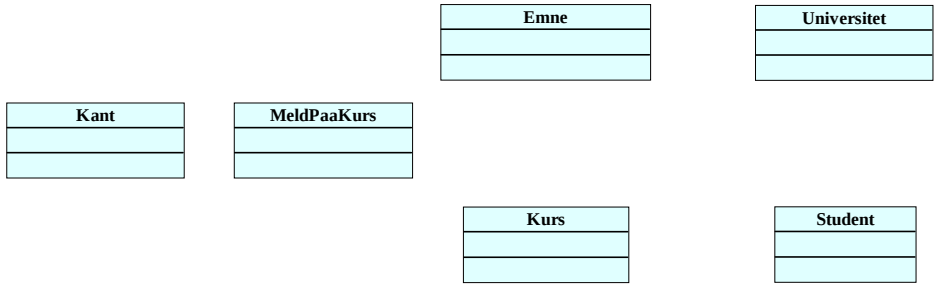
Dagens forelesning

- Kort repetisjon
  - Objektorientert modellering (oblig 2)
    - Notasjon: UML klassediagram og objektdiagram
    - Metode: Fra sekvensdiagram til klassediagram
- Design av persistens
  - Relasjonsdatabaser (tabelldatabaser)
    - Datamodellering med UML
    - Object-Relational (O-R) Mapping
    - Structured Query Language (SQL)
    - Eksempel: Hibernate
  - Objektorienterte databaser
    - Eksempel: ObjectStore

Normal hendelsesflyt for ”Meld på kurs”

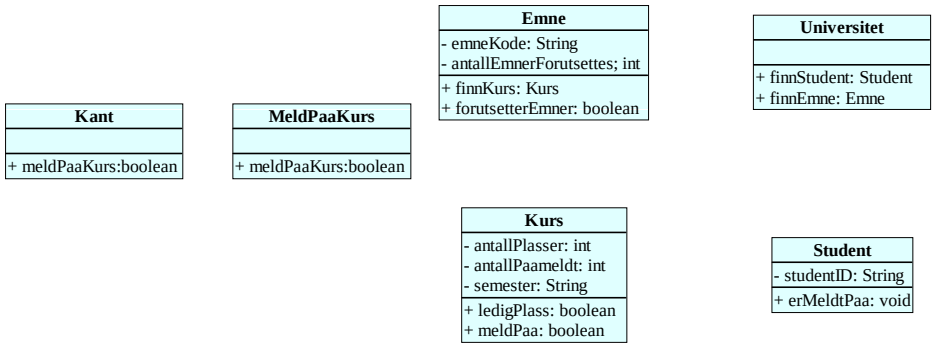


Klasser involvert i hovedflyt



1) Inkluder klassene du finner i sekvensdiagrammet

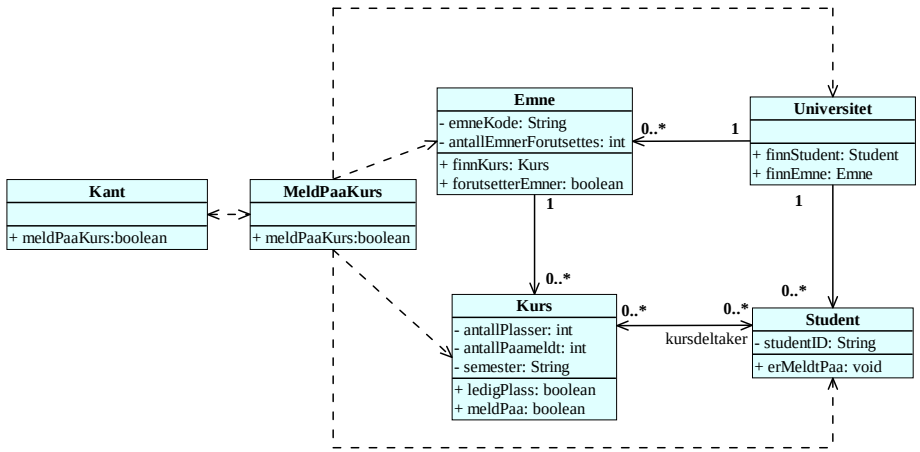
Metoder og attributter for hovedflyt



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

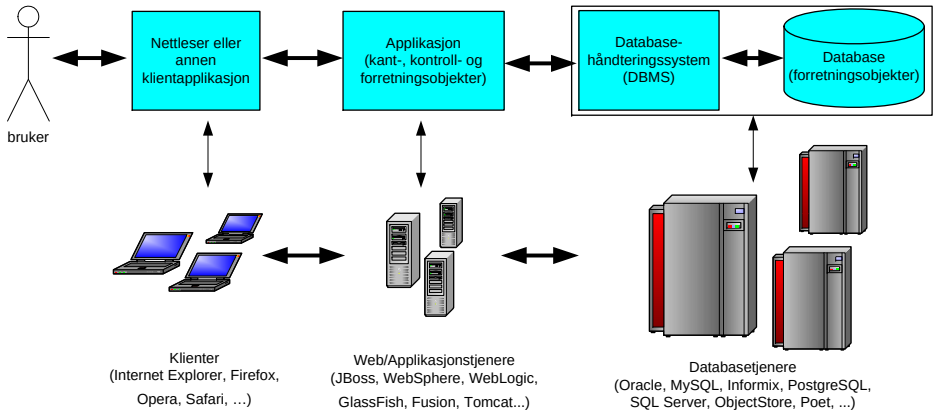
Metode for ansvarsdrevet OO med UML

- **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**
  - **Detaljert design (persistens, arkitektur, brukergrensesnitt, spesifisere tester)**
  - **…**

Persistens

- Forretningsobjektene i vår applikasjon må lagres på en eller annen måte
  - *Utenfor* applikasjonens minne, på et permanent lagringsmedium
- Til dette formålet bruker vi en *database*
  - typisk er databasen implementert i et dedikert databasehåndteringssystem (Data Base Management System - DBMS)
  - Men for svært enkle systemer holder det noen ganger med "hjemmesnekrede løsninger", som for eksempel at hvert objekt lagres og hentes direkte som *filer* av vår applikasjon (dvs, vi bruker operativsystemets filhåndteringssystem)
- Forretningsobjektene i vår applikasjon lagres i databasen, og hentes derfra når applikasjonen trenger dem
  - I eksempelet vårt: Emne, Kurs, Universitet, Student

Eksempel trelagsarkitektur



Hva er et databasehånderingssystem?

- Tilbyr grensesnitt for applikasjoner
  - API for programmerere (for eksempel *JDBC* mot relasjonsdatabaser)

  (har ofte også egne brukergrensesnitt for å gjøre direkte spørringer mot databasen)
- Utfører (og optimaliserer) spørringer og oppdateringer
  - brukerdata
  - metadata (data om brukerdata)
- Håndhever skranker/integritetsregler
  - Eks 1: "Kurs må være assosiert med nøyaktig ett emne".
  - Eks 2: "Et emne kan ikke slettes hvis det holdes kurs i emnet" (dvs, kursene må slettes først)
- Håndterer flere brukere samtidig

  (gjelder ikke enbruker-DBMS)
- Gjennomfører oppdateringer av data som *transaksjoner*
- Utøver tilgangskontroll
- Sikrer data

To hovedtyper av databaser

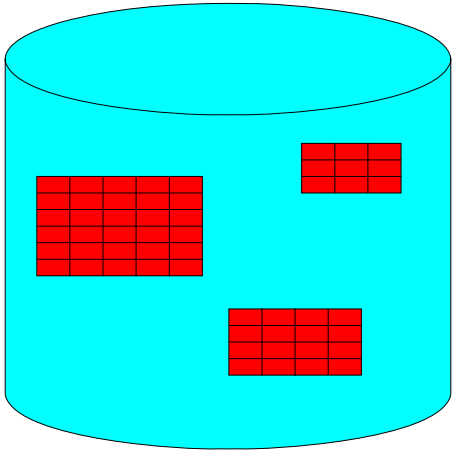
- Relasjonsdatabaser (tabelldatabaser)
  - Data lagres som tabeller (relasjoner) med forekomster
  - Hvis vi skal bruke en relasjonsdatabase som lagringsmedie må forretningsobjekter og assosiasjoner mellom disse oversettes (*mappes)* til/fra relasjoner (Object-Relational Mapping – O/RM)
- Objektorienterte databaser
  - Lagrer og henter objekter som brukes i en applikasjon uten at du trenger å bry deg om hvordan (stort sett)
  - "Forstår" objektorienterte konstruksjoner (assosiasjoner som for eksempel er implementert som en HashMap, arv, osv)
- Det finnes også hybridløsninger (object-relational)

Relasjoner og relasjonsdatabaser

- **Relasjon (litt forenklet)**
  - **Et matematisk begrep som kan tolkes som en tabell med verdier *der alle linjer i tabellen er forskjellige fra hverandre:***
    - ***Relasjonen* har et *entydig navn***
    - ***Relasjonen består av en rekke attributter***
    - ***Attributter* har et *entydig navn* innen relasjonen**
    - ***Attributtenes rekkefølge* skal være uten betydning**
    - **Attributtene er *atomiske* (ikke sammensatte strukturer)**
- **Relasjonsdatabase**
  - **En samling relasjoner**

- ***E. F. Codd: "A Relational Model for Large Shared Data Banks",*** *Communications of the ACM, Vol 13, Number 6 (June 1970)*

Relasjonsdatabasen



- **En relasjonsdatabase består av tabeller (relasjoner).**
- **Hver linje representerer en gitt entitet (forekomst), og har en unik identitet definert ved en primærnøkkel (primary key).**
- **Hver celle i tabellen inneholder verdien til et av attributtene til entiteten**
- **Tabellene assosieres med hverandre via fremmednøkler (foreign keys)**

Eksempel på tabeller

Ansatt

ansattID {pk}	navn	pID {fk}{null}
1	Per	
2	Kari	2
3	Ola	4

Parkeringsplass

pID {pk}	beliggenhet
1	P4
2	P3
3	P3
4	P1

Primærnøkler og fremmednøkler

- **Hver tabell må ha en primærnøkkel (primary key), som unikt identifiserer hver enkelt forekomst i tabellen**
  - **Er vanligvis ett attributt, men *kan* også være sammensatt av flere enn ett attributt (som til sammen blir en primærnøkkel)**
- **En tabell kan ha null, en eller flere fremmednøkler (foreign keys). En verdi i en *fremmednøkkel* har en tilsvarende verdi i en *primærnøkkel* i en annen tabell.**
  - **Dvs, en fremmednøkkel relaterer en gitt forekomst med forekomster i andre tabeller.**
  - **UML assosiasjoner realiseres som fremmednøkler i en relasjonsdatabase**

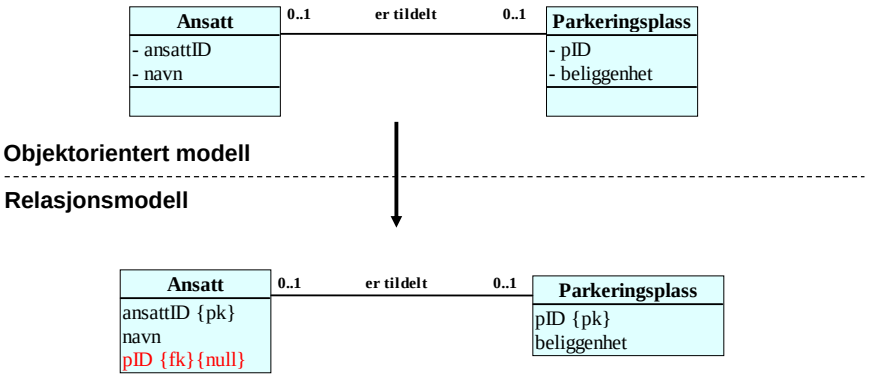
Object-relational mapping: fra objekter til relasjoner

- For hver objektorienterte forretningsklasse (entity objects)
  - lag en tilsvarende relasjon (tabell) som inneholder attributtene fra klassen samt en ny primærnøkkel dersom en passende identifikator ikke allerede er definert i klassen
- For en-til-en assosiasjoner
  - Inkluder en fremmednøkkel i *en* av klassene (valgfritt hvilken) som tilsvarer den andre klassens primærnøkkel
- For en-til-mange assosiasjoner
  - Inkluder en fremmednøkkel i klassen på *mange-siden* av assosiasjonen som tilsvarer *en-sidens* primærnøkkel
- For mange-til-mange assosiasjoner
  - Opphøyes til å bli en egen relasjon (tabell): Lag en ny tabell som inneholder to fremmednøkler, en for hver primærnøkkel i de to klassene i assosiasjonen

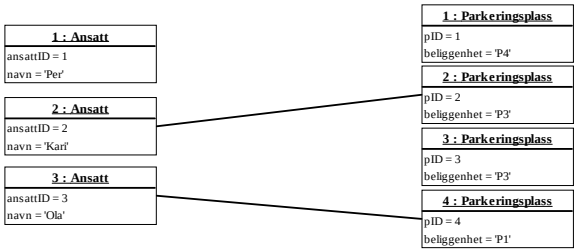
UML klassediagrammer kan brukes til å definere en datamodell

- En datamodell (relasjonsmodell) inneholder ikke metoder, men inneholder primærnøkler og evt. fremmednøkler (som realiserer assosiasjoner)
- Nøkkelordet *{fk}* definerer at attributtet er en fremmednøkkel
- Nøkkelordet *{pk}* definerer at et attributt er en (del av en) primærnøkkel
- Nøkkelordet *{null}* definerer at et attributt kan være tomt, ellers antas *{not null}*

En-til-en assosiasjon



Objekter og tilsvarende forekomster



NB! Objektidentifikatorene er her satt til samme verdier som primærnøklene for å tydeliggjøre sammenhengen.

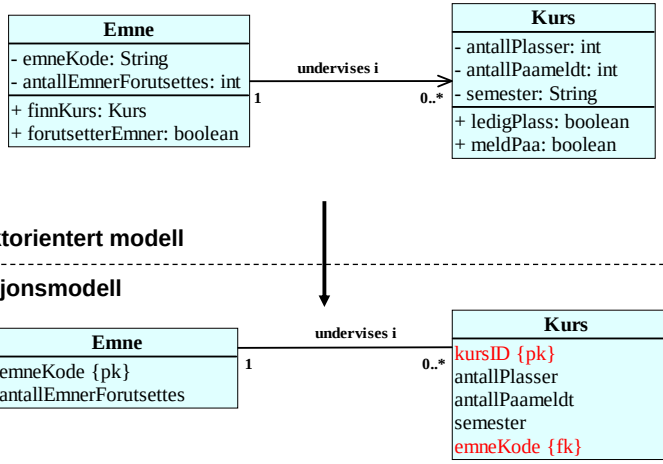
Ansatt

ansattID {pk}	navn	pID {fk}{null}
1	Per	
2	Kari	2
3	Ola	4

Parkeringsplass

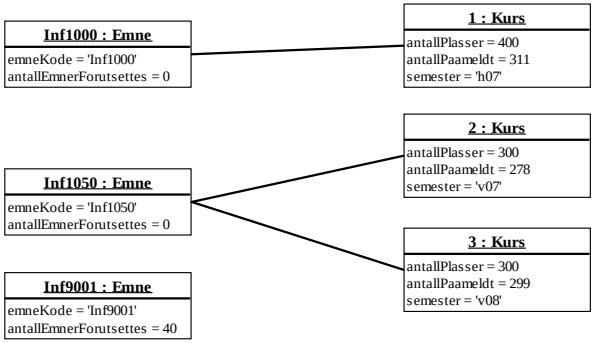
pID {pk}	beliggenhet
1	P4
2	P3
3	P3
4	P1

En-til-mange assosiasjon



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

Objekter og tilsvarende forekomster



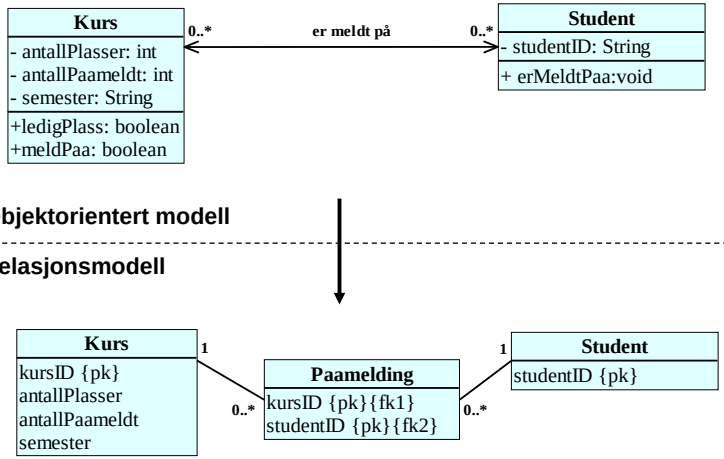
Emne

emneKode {pk}	antallEmner Forutsettes
Inf1000	0
Inf1050	0
Inf9001	40

Kurs

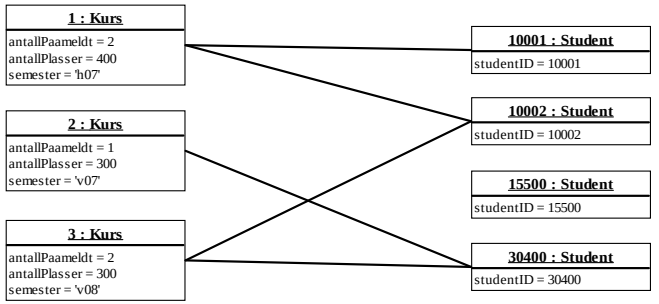
kursID {pk}	antallPlasser	antallPaameldt	semester	emneKode {fk}
1	400	311	h07	Inf1000
2	300	278	v07	Inf1050
3	300	299	v08	Inf1050

Mange-til-mange assosiasjoner



Assosiasjonen opphøyes til en egen entitet i en datamodell

Objekter og tilsvarende forekomster



Kurs

kursID {pk}	antallPlasser	antallPaameldt	semester
1	400	2	h07
2	300	1	v07
3	300	2	v08

Paamelding

kursID{pk}{fk1}	studentID {pk}{fk}
1	10001
1	10002
2	30400
3	10002
3	30400

Student

studentID{pk}
10001
10002
15500
30400

Structured Query Language (SQL)

SQL er et standard programmeringsspråk for å håndtere *tabelldatabaser*.

- For forekomstmanipulering (DML –"Data Manipulation Language"):
  - SELECT
  - INSERT
  - UPDATE
  - DELETE
- For skjemamanipulering (DDL –"Data Definition Language")
  - CREATE
  - ALTER
  - DROP
- For transaksjons- og tilgangskontroll (DCL –"Data Control Language")
  - COMMIT
  - ROLLBACK
  - GRANT
  - REVOKE

Eksempel på SELECT

SELECT <attributene som skal være med>

kursID, semester, antallPlasser

FROM <tabellen(e) som inneholder dataene>

Kurs

WHERE <utvalgskriterier>

emneKode = 'Inf1050' AND semester ='v08'

Eksempel på INSERT

INSERT INTO

Kurs (kursID, emneKode, semester, antallPlasser, antallPaameldt)

VALUES

(4, 'Inf1050','v09', 300, 0)

Eksempel på UPDATE

Alt 1) UPDATE Kurs SET antallPaameldt = 1

WHERE kursID = 4

- Antall påmeldte til kurset settes lik 1

Alt 2) UPDATE Kurs

SET antallPaameldt = antallPaameldt + 1

WHERE emneKode = 'inf1050' AND semester = 'v09'

- Antall påmeldte til kurset økes med 1

Eksempel på DELETE

DELETE FROM Kurs WHERE kursID = 4

eller…

DELETE FROM Kurs WHERE emneKode = 'inf1050'
AND semester = 'v09'

- Fjerner kurset for Inf1050 v09.

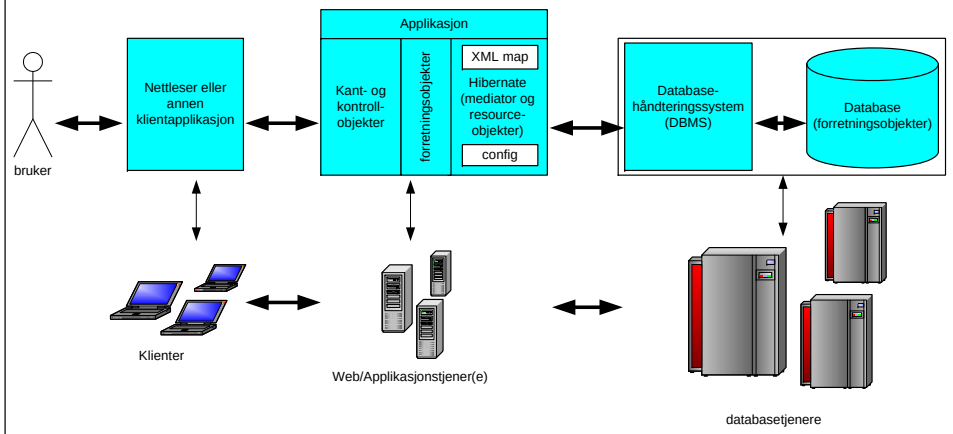
DELETE FROM Kurs WHERE emneKode = 'Inf1050'

- Fjerner alle kurs for Inf1050!

Transaksjoner

- **En transaksjon grupperer SQL-kommandoer inn i et udelelig stykke arbeid slik at enten *alle* eller *ingen* av endringene blir permanente.**
- **ROLLBACK: Reverserer alle SQL-kommandoer som er utført så langt, slik at databasen ser ut som ved forrige COMMIT**
  - **Kalles typisk når feilsituasjoner oppstår i løpet av en transaksjon**
- **COMMIT: Endringene som et resultat av SQL-kommandoene siden forrige COMMIT blir gjort permanente.**

Arkitektur med Hibernate for persistens vha relasjonsdatabaser

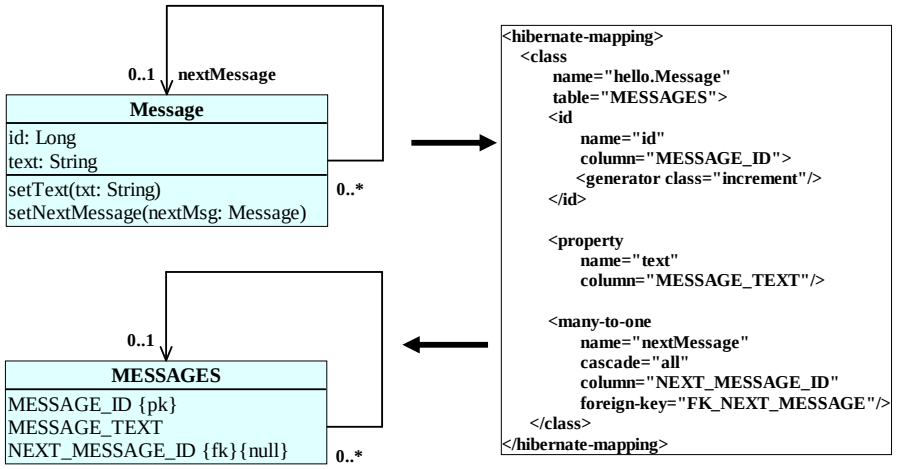


O-R mapping med Hibernate

- **Støtter oversetting (mapping) fra objektorienterte klassemodeller til relasjonsdatabaser. Du kan selv skrive XML mapping-filer eller bruke automatiske verktøy (for eks. XDoclet).**
  - **Har endel regler for hvordan assosiasjoner bør mappes slik at navigering over assosiasjonene blir effektive**
  - **Kan generere relasjonsdatabaseskjemaet fra XML mapping-filene**

  (XML = Extensible Markup Language, brukes til å beskrive data og "data om data", dvs metadata)
- **Transparent å opprette, finne, oppdatere og slette objekter i databasen**
  - **Hibernate bruker en XML mapping-fil sammen med *reflection* for å finne ut hvordan dette skal gjøres**
  - **Støtter transaksjoner, "lazy load" og "dirty checking" (se RASD, kap 8.4)**
- **Persistens ved transitivitet (via assosiasjoner)**
  - **Støtter standard Java Collections: Hibernate vet om du endrer noe i et av objektene i en Collection og vil automatisk gjøre SQL Update eller SQL Insert**
- **Kan bruke både SQL og objektorientert Hibernate Query Language (HQL)**
  - **Men det ALLER meste av SQL genereres automatisk**
- **For detaljer, se http://www.hibernate.org**

XML-mapping for Message

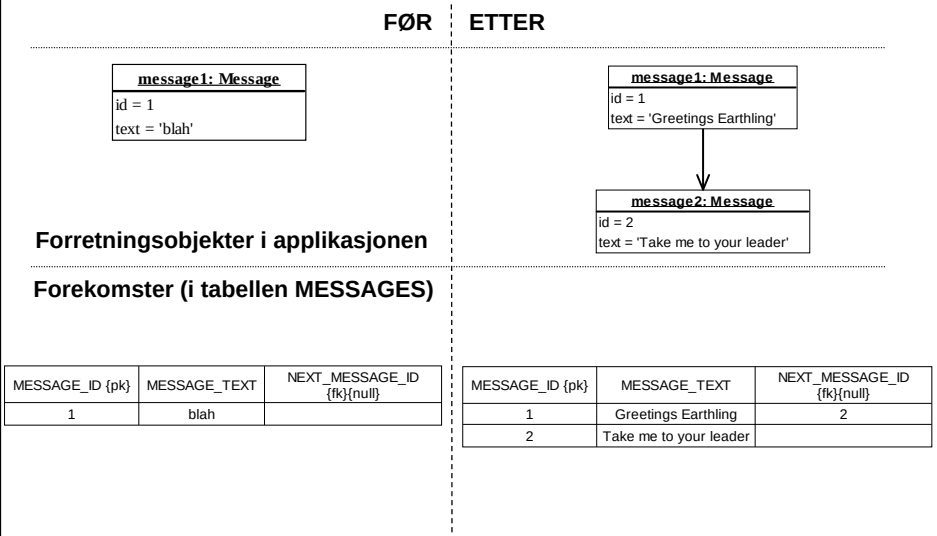



```
<hibernate-mapping>
  <class
      name="hello.Message"
      table="MESSAGES">
      <id
          name="id"
          column="MESSAGE_ID">
          <generator class="increment"/>
      </id>

      <property
          name="text"
          column="MESSAGE_TEXT"/>

      <many-to-one
          name="nextMessage"
          cascade="all"
          column="NEXT_MESSAGE_ID"
          foreign-key="FK_NEXT_MESSAGE"/>
  </class>
</hibernate-mapping>
```

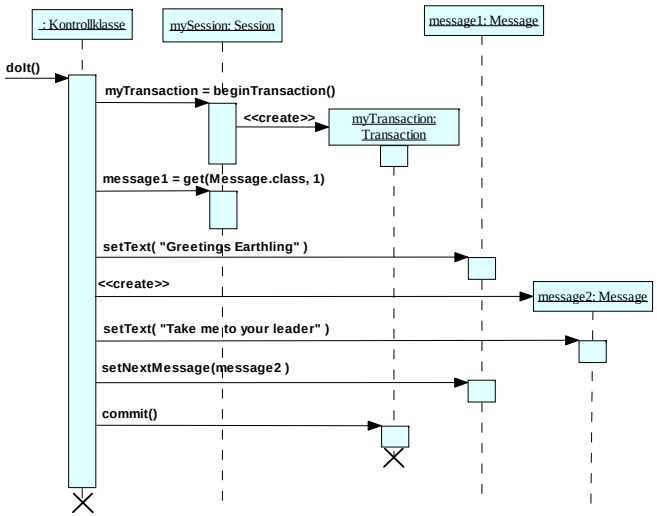

Vårt scenario



MESSAGE_ID {pk}	MESSAGE_TEXT	NEXT_MESSAGE_ID {fk}{null}
1	blah	

MESSAGE_ID {pk}	MESSAGE_TEXT	NEXT_MESSAGE_ID {fk}{null}
1	Greetings Earthling	2
2	Take me to your leader	

Eksempel på persistens med Hibernate



Hibernate vil da utføre følgende SQL-kommandoer

- select MESSAGE_ID, MESSAGE_TEXT, NEXT_MESSAGE_ID from MESSAGES where MESSAGE_ID = 1
- insert into MESSAGES (MESSAGE_ID, MESSAGE_TEXT, NEXT_MESSAGE_ID) values (2, 'Take me to your leader', null)
- update MESSAGES set MESSAGE_TEXT = 'Greetings Earthling', NEXT_MESSAGE_ID = 2 where MESSAGE_ID = 1

ObjectStore: eksempel på OO database

- ObjectStore er en objektorientert database som gir en helt sømløs (transparent) måte å lagre forretningsobjektene på.
- Objectstore vs Hibernate:
  - ObjectStore: Ingen O-R mapping (eller SQL) er nødvendig, fordi objektene lagres som de er.
  - ObjectStore konverterer klassene til noe de kaller “persistence capable” og “persistence aware”. Man må også bruke egne collection klasser (for eksempel: OSHashMap istedet for standard HashMap)
  - For å få tilgang til persistente objekter bruker man et eller flere “rotobjekter”, og fra disse kan man få tilgang til de andre objektene ved å følge pekere (assosiasjoner).
    - KursRegistrering: Universitet ville vært et bra rotobjekt
  - Bortsett fra det er en applikasjon som bruker Objectstore ganske lik med en som bruker Hibernate
    - Tilsvarende funksjonalitet for transaksjonshåndtering, støtter automatisk lazy load, dirty checking, transitiv persistens, osv.

Hvorfor brukes objektorienterte databaser så lite?

- Data blir applikasjonsspesifikke
  - Data varer evig, mens applikasjoner kommer og går
- Ytelse?
- Portabilitet
  - Lettere å bytte relasjonsdatabase enn å bytte ut en objektorientert database (?)
- Lite marked => større risiko
- Support
  - Relasjonsdatabaser har masse gratis programvare…
- Hibernate gjør omtrent det samme!
  - Og du har i tillegg muligheten til å bruke mer avanserte funksjoner som tilbys av relasjonsdatabaser