# Use case drevet design med UML

**Bente Anda**

**26.09.2005**



# I dag

- Domenemodeller
- System sekvensdiagrammer
- Operasjonskontrakter
- GRASP patterns
- Designmodeller med sekvens- og klassediagram

## Domenemodell

- Domenemodellen brukes i objekt-orientert analyse
- Domenemodellen viser konsepter i applikasjonsdomenet, konseptuelle klasser, og forholdet mellom dem: ideer, ting, objekter.
- Hensikten med domenemodellen er å forstå konseptene og få en oversikt over terminologi.



## Domenemodell forts.

- Domenemodellen beskrives med UML klassediagrammer uten metoder, og utarbeides gjennom flere iterasjoner.
- Use case modell og domenemodell utformes parallelt.
- Domenemodellen fanger opp informasjonen om entiteter som er beskrevet i use casene.
- Use casene presiseres ved utforming av domenemodellen.
- Domenemodellen er et viktig verktøy for å sjekke at use casene er beskrevet med riktig detaljeringsnivå.
- Det er bedre å spesifisere for mange konseptuelle klasser enn for få.

## Hvordan finne domeneklasser?

- Lag en liste over kandidater til klasser:

2 forskjellige tilnærminger:

  - Lag en liste over kandidater til domeneklasser (for eksempel basert på liste s.140-141)
  - Finn substantiver og substantivutrykk i use case beskrivelsene

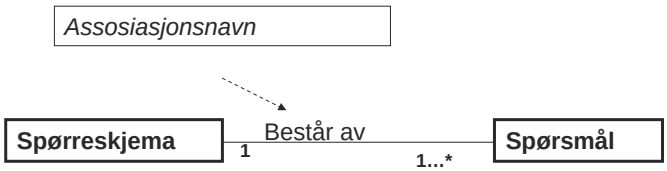
- Ta bort unødvendige klasser
  - Noen klasser kan vise seg å være attributter.



## Assosiasjoner

- En assosiasjon er en relasjon mellom to klasser som viser at det er en sammenheng mellom dem.
  - En assosiasjon kan ha et navn
- Brukes i domenemodellen hvis informasjon om relasjonen skal lagres.
- Hver ende av assosiasjonen kalles en rolle, en rolle kan ha:
  - Multiplisitet
  - Navn
  - Navigasjon

# Eksempel




Assosiasjonen går i begge retninger.

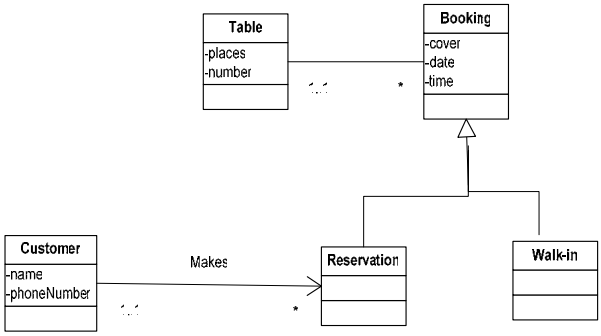
Liste over mulige assosiasjoner på s. 155



# Attributter

- Inkluder de attributter som det må lagres informasjon om i følge use casene
- Attributter er vanligvis datatyper, f.eks int, Boolean, string, dato
- Modeller konsepter som klasser, ikke attributter. Hvis det er tvil, lag en klasse

## Domenemodell for restaurantsystemet






## System sekvensdiagrammer

- System sekvensdiagrammer visualiserer operasjonene som genereres av aktørene.
- System sekvensdiagrammene identifiserer nødvendig input til use casene (brukergrensesnitt, kommunikasjon med andre systemer)
- Input til konstruksjon av system sekvensdiagrammene er domenemodellen (utgangspunkt for klasser) og use case.

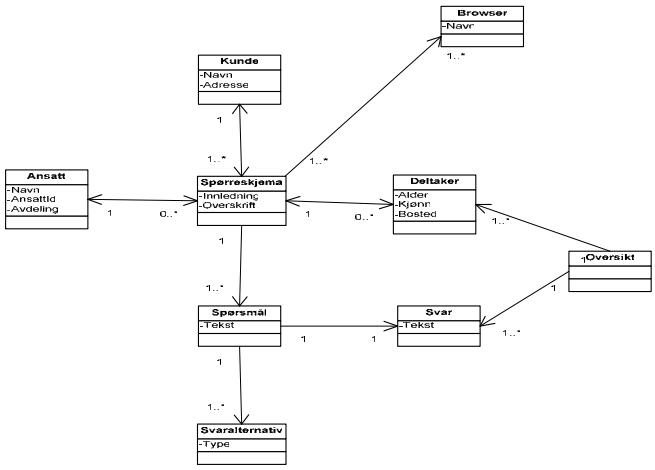
# Eksempel: Spørreskjemagenerator

Kravspesifikasjon:

Et meningsmålingsinstitutt ønsker å få laget et system der spørreskjema er på Internett/Web. Systemet skal gjøre det enkelt å legge et spørreskjema ut på Web, enkelt for andre å fylle ut skjemaene på Web. Svarene skal lagres på et format som kan eksporteres til andre verktøy (f eks "strukturert tekst" som kan importeres til et regneark). Deltakerne skal kunne lagre svarene underveis og fortsette utfyllingen av skjemaet senere. Til noen av spørsmålene er det nødvendig å lese en del tekst. Meningsmålingsinstituttet ønsker å ha en enkel oversikt over svarene som er kommet inn, f eks hvor mange som har svart på de ulike spørsmålene. Det som skal lages er altså ikke et enkelt spørreskjema på Web, men en "spørreskjema-generator" for Web.



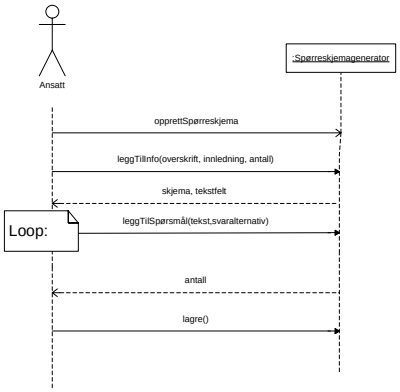
# Domenemodell med assosiasjoner



# Use case og system sekvensdiagram

Use case 'Generer spørreskjema'

1. Ansatt ber om å få opprette nytt spørreskjema
2. Systemet ber om overskrift, innledning og antall spørsmål som spørreskjemaet skal bestå av.
3. Ansatt skriver inn nødvendig informasjon
4. Systemet sjekker at alle felt er utfylt
5. Systemet viser et spørreskjema der tekst til spørsmål skal fylles inn.
6. Ansatt skriver inn tekst og evnt. svaralternativ til hvert av spørsmålene
7. Systemet sjekker at riktig antall spørsmål har fått tekst
8. Ansatt ber om at spørreskjema blir lagret
9. Systemet lagrer spørreskjemaet






# Operasjonskontrakter

- Kontrakter med pre- og post betingelser på operasjoner i use casene for å gjøre use casene mer presise.
- Prebetingelser beskriver forutsetningene for å gjennomføre operasjonen.
- Postbetingelser beskriver endringer i tilstandene til objektene i domene modellen.

## Eksempel på kontrakt

- Operasjon: leggTilInfo(overskrift, innledning, antall)
- Kryssreferanse: Use case 'Generer spørreskjema'
- Prebetingelse: Et nytt spørreskjema er opprettet
- Postbetingelser:
  - spørreskjema.overskrift er blitt overskrift
  - spørreskjema.innledning er blitt innledning
  - antall spørsmål er opprettet og assosiert med spørreskjema



## Sekvensdiagrammer

- Interaksjonsdiagrammer er det fremste verktøyet for tilordning av ansvar til objektene
- Vi fokuserer her på sekvensdiagrammer
- For hvert use case lages et sekvensdiagram for normal hendelsesflyt og komplekse og hyppig forekommende variasjoner
- Stegene i use casene vises som meldinger som sendes mellom objektene
- Sekvensdiagrammene viser hvordan objekter kommuniserer ved hjelp av meldinger (metodekall)
- Sekvensdiagrammene lages parallelt med klassediagrammer

# Forts. sekvensdiagrammer: fra krav til design

Et UML sekvensdiagram

- viser hendelsesflyten i et use case
- viser **interaksjoner** (samarbeid) mellom objekter i systemet
- viser rekkefølgen på meldingene (abstrakt) som sendes mellom objektene
- kan brukes til å identifisere **metodene** (konkret) til objektene i systemet



# Objekt design

- Hvilke er de riktige objektene?
- Hva er ansvarsområdene til objektene, hva skal de vite og hva skal de gjøre?
- Hvem samarbeider objektene med?
- Objektene i er av typene kant-, kontrol- og entitetsobjekter
  - Entitetsobjekter holder data
    - Modellerer varig informasjon og assosiasjoner
    - Ofte abstraksjoner av entiteter i den virkelige verden (domene-klasser)
    - Kan finnes ved å lete etter substantiver i kravspesifikasjoner
  - Kantobjekter kommuniserer med omverdenen
    - Modellerer systemets grensesnitt
  - Kontrollobjekter kontrollerer interaksjonene i hele systemet eller i et use case
    - Modellerer oppførsel til et eller flere use case
    - Håndterer kontrollflyten
- Merk også at en klasse betyr forskjellige ting avhengig av perspektiv

# Objektdesign: Ansvarstilordning

- Ansvar er knyttet til objektet i form av dets oppførsel
  - *Handling:* Opprette objekt, beregning, initiere handlinger i andre objekter, kontrollere og koordinere handlinger i andre objekter.
  - *Kunnskap:* Vite om private data, vite om relaterte objekter, vite om ting som det kan utlede eller beregne
- Ansvar er ikke det samme som metoder, men metoder implementeres for å oppfylle ansvaret



# Kjennetegn på ’god’ design

- En god utforming gjør den jobben den er ment å gjøre
- En god utforming er enkel og elegant

  Eleganse innebærer å finne akkurat riktig abstraksjonsnivå
- En god utforming er gjenbrukbar, utvidbar og enkel å forstå
- Et godt objekt har et lite og veldefinert ansvarsområde
- Et godt objekt skjuler implementasjons-detaljer fra andre objekter - *Grady Booch*

## Modularisering

- Høy kohesjon
  - Et objekt skal bare ha ansvar for relaterte ting
- Lav kobling
  - Et objekt skal ha samarbeid med et begrenset antall andre objekter



## Høy kohesjon

- Kohesjon er et mål på hva slags ansvar et objekt har og hvor fokusert ansvaret er
- Et objekt som har moderat ansvar og utfører et begrenset antall oppgaver innenfor ett funksjonelt område har høy kohesjon
- Objekter med lav kohesjon har ansvar for mange ting innen ulike funksjonelle områder

## Lav kobling

- Kobling er et mål på hvor sterkt et objekt er knyttet til andre objekter
- Et objekt med sterk kobling er avhengig av mange andre objekter noe som kan gjøre endring vanskelig



## Patterns

- Patterns  er navngitte retningslinjer for hvordan ansvar skal fordeles i ulike situasjoner.
- Patterns representerer sammenstilling av erfaring
- Patterns brukes i prosessen med å forfine sekvensdiagrammer
- GRASP – 'Patterns of General Principles in Assigning Responsibilites' =
  Mønster for problemløsning
- Sentrale patterns er Ekspertprinsippet, Kontrollobjektprinsippet og Skaperprinsippet

# Ekspertprinsippet (Information Expert)

- Problem: Hva er det generelle prinsipp for å tilordne ansvar til objekter?
- Løsning: La det objektet som har kunnskapen (dataene) også behandle den.

- Hvordan:
  - Begynn med å formulere ansvarsområdet:
    *Hvilket objekt har ansvar for å vite om det totale antall svar på undersøkelsen?*
    *Hvilket objekt har ansvar for å ha oversikt over alle bookingene?*
  - Se i domenemodellen etter en klasse som kan brukes eller utvides til å lage en designklasse.
    *Det finnes en klasse 'Oversikt' i domenemodellen, lag en designklasse 'Oversikt' og gi den ansvaret for kunnskap om antall svar med metoden 'visAntallSvar()'.*
    *Det finnes ingen klasse som har oversikt over alle bookingene, lag en designklasse 'Restaurant' og gi den ansvaret for dette*.

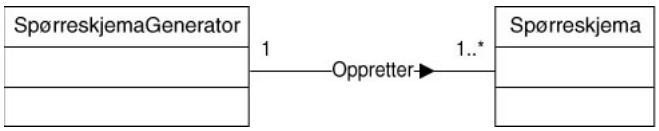


# Skaperprinsippet (Creator)

- Problem: Hvem er ansvarlig for å opprette nye objekter?
- Løsning: La det objektet som må vite om de nye objektene lage dem
- Hvordan: Gi klasse B ansvaret for å opprette et objekt av klasse A dersom ett av følgende er sant:
  - B inneholder A-objekter
  - B registrerer A-objekter
  - B bruker A-objekter
  - B har data som sendes til A-objektet når det opprettes

# Forts. skaperprinsippet

- Se i domenemodellen etter mulige klasser
- *Designklassen 'Restaurant' oppretter bookinger*
- *Vi trenger en 'SpørreskjemaGenerator' klasse for å opprette nye skjemaer*

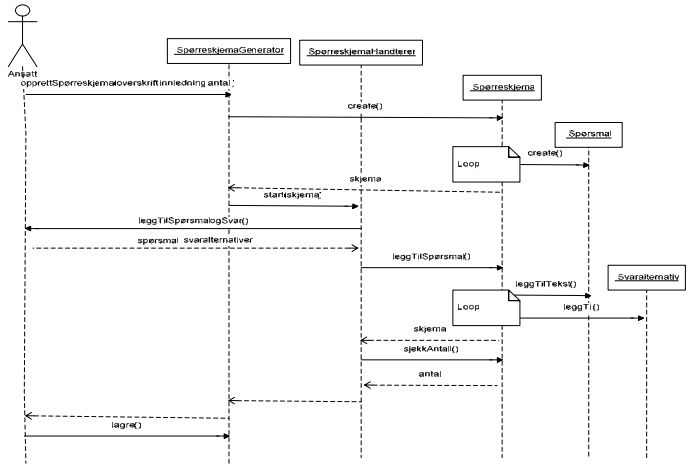





# Kontrollobjektprinsippet (Controller)

- Problem Hvem er ansvarlig for å håndtere systemhendelser (en hendelse som genereres av en ekstern aktør)?
- Løsning: Tilordne ansvar for å håndtere en systemhendelse til en av følgende klasser:
  - En klasse som representerer systemet eller subsystemet (fasadekontroller)
  - En klasse som representerer et use case scenario der systemhendelsen forkommer ofte
    Navn:
    *'SpørreskjemaHåndterer' håndterer use caset 'Generer spørreskjema'*
    *'Bookingsystem' håndterer alle use casene i restaurantsystemet*
- Kontrollobjekter delegerer oppgaver til andre objekter

# Generer spørreskjema






# Designmodellen

- Lag design-klassediagram parallelt med sekvensdiagrammer
- Lag noen sekvensdiagrammer, oppdater klassediagrammet, utvid sekvensdiagrammet etc.
- Designklassene er systemklasser, ikke konseptuelle klasser som i domenemodellen

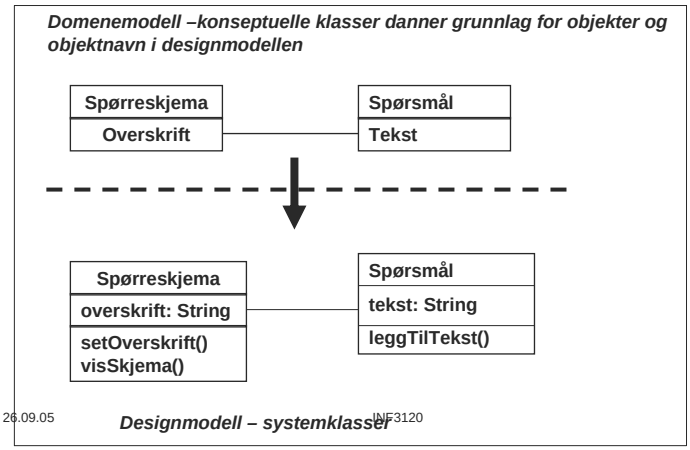
# Framgangsmåte for designmodellering

- Identifiser klasser ved å gå gjennom alle sekvensdiagrammene
- Klassenavnene er inspirert av navn i domenemodellen
- Legg til metodenavn ved å analysere sekvensdiagrammene
  Eks: Meldingen leggTilTekst() sendes til Spørsmåls-objektet. Objektet må derfor inneholde en leggTilTekst() metode

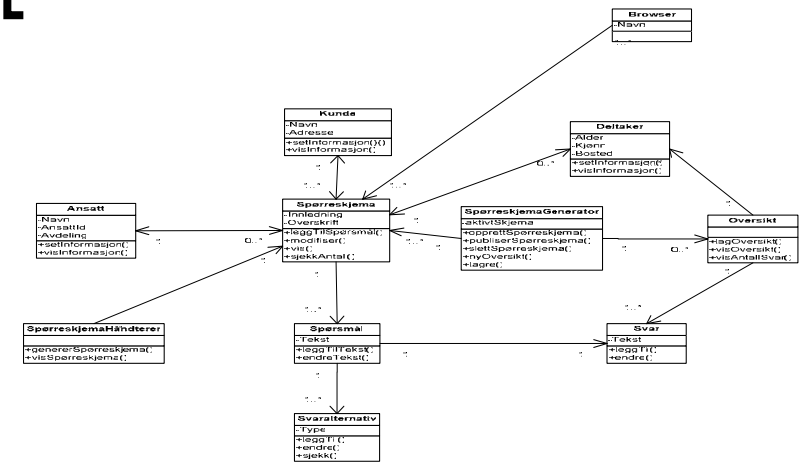


# Fra domenemodell til designmodell

Noen av objektene som kommuniserer via meldinger hentes fra domenemodellen



# Designmodell for spørreskjemagenerator






# Ansvarsfordeling

- **Spørreskjemagenerator:** Lager nye spørreskjemaer, publiserer dem og viser oversikter.
- **Spørreskjema:** Kjenner sin egen overskrift, innledning og sine spørsmål
- **Spørsmål:** Kjenner sine egne tekster og kan modifisere disse
- **Svar:**Kjenner sitt eget svar og kan modifisere dette
- **SpørreskjemaHåndterer:** Kontrollobjekt som koordinerer objektene
- **Svaralternativer:** Kjenner lovlige svar på et gitt spørsmål.
- **Oversikt:** Vet om innholdet i besvarelsene og lager statistikk

## Oppgave

- Utvid domenemodellen til en designmodell for restaurantsystemet



## Oppsummering

- I objektorientert analyse utarbeides use case modell og domenemodell i parallell.
- System sekvensdiagrammer visualiserer interaksjonen mellom aktør og system og nødvendig input til systemet.
- Operasjonskontrakter kan gjøre use casene mer presise.
- Scenariene i et use case realiseres gjennom objekter som samarbeider. Dette illustreres i sekvensdiagrammer.
- Klassediagrammer og sekvensdiagrammer utarbeides i parallell, designklasser og metoder finnes ofte fra sekvens-diagrammene
- Bruk av patterns kan lette og forbedre designarbeidet

## Løsning oppgave 5 (22/9) - 'Record arrival (Registrer ankomst)' og 'Record walk-in (Registrer ankomst uten reservasjon)'

| | |
|---|---|
| **Actor** | Head waiter |
| **Extension points** | Walk-in customer, step 2 |
| **Main success scenario** | 1. The head waiter performs the "Display bookings" use case<br>2. The head waiter confirms arrival for a selected booking<br>3. The system records this and updates the display, marking the customer as having arrived |

| | |
|---|---|
| **Actor** | Head waiter |
| **Extension points** | None |
| **Main success scenario** | 1. The head waiter creates a walk-in booking by entering the time of the booking, the number of covers and the table number<br>2. The system records and displays the new booking |



## Forts. løsning oppgave 5 -'Transfer table (Bytt bord)'

| | |
|---|---|
| **Actor** | Head waiter |
| **Extension points** | None |
| **Main success scenario** | 1. The head waiter performs the "Display bookings" use case<br>2. The head waiter changes the table allocation of the booking<br>3. The system records the alteration and updates the display |