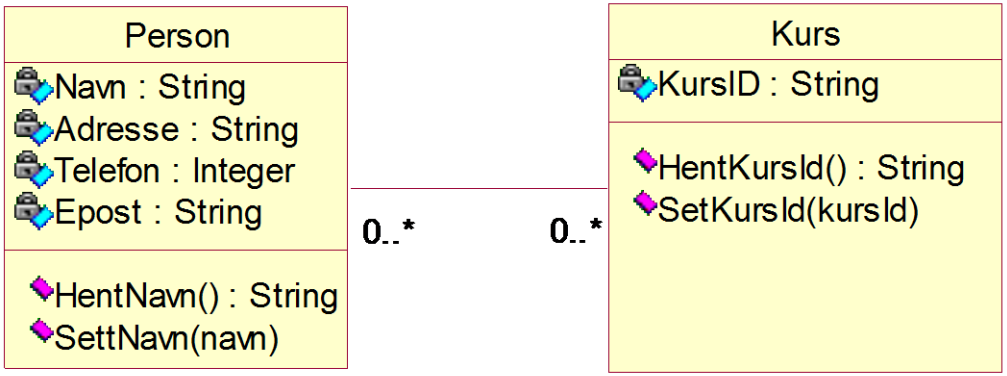
Systemutvikling
Mer om objektorientering og UML

HØGSKOLEN I OSLO
OG AKERSHUS

Høgskolen i Oslo og Akershus
Avdeling for ingeniørutdanning
15 september 2014

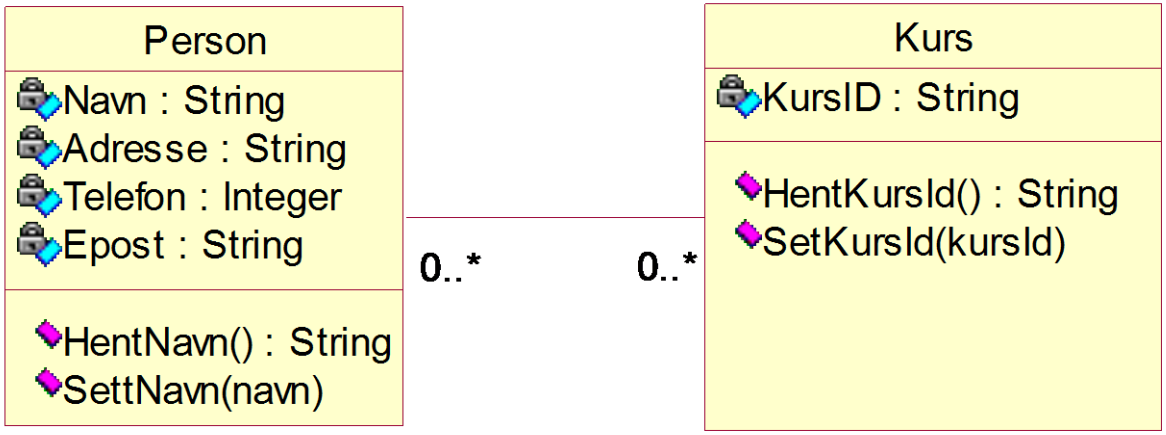
Yngve Lindsjørn- yngve.lindsjorn@hioa.no



Temaer i dagens forelesning

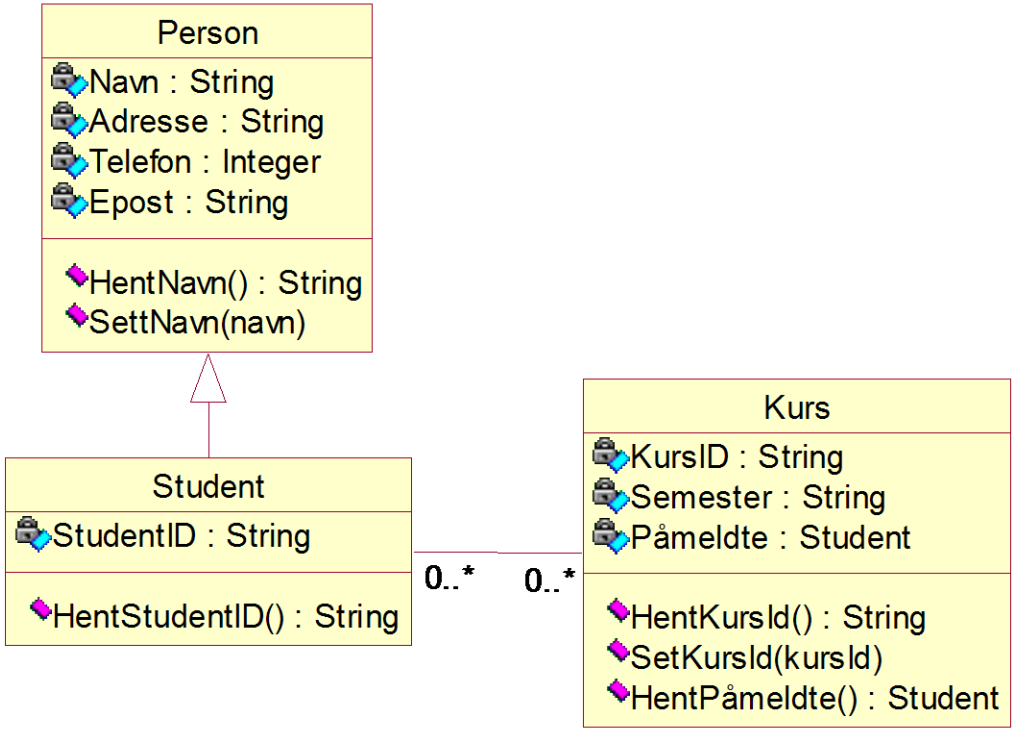
- Objektorientert design
  - Objektdesign og ansvarstilordning
  - Bruk av UML
    - Fokus på klassediagrammer
- Design og implementering
- Designmodeller
- Designmønstre (”Design Patterns”)
- Oppsummering

Klassediagram – Person tar kurs



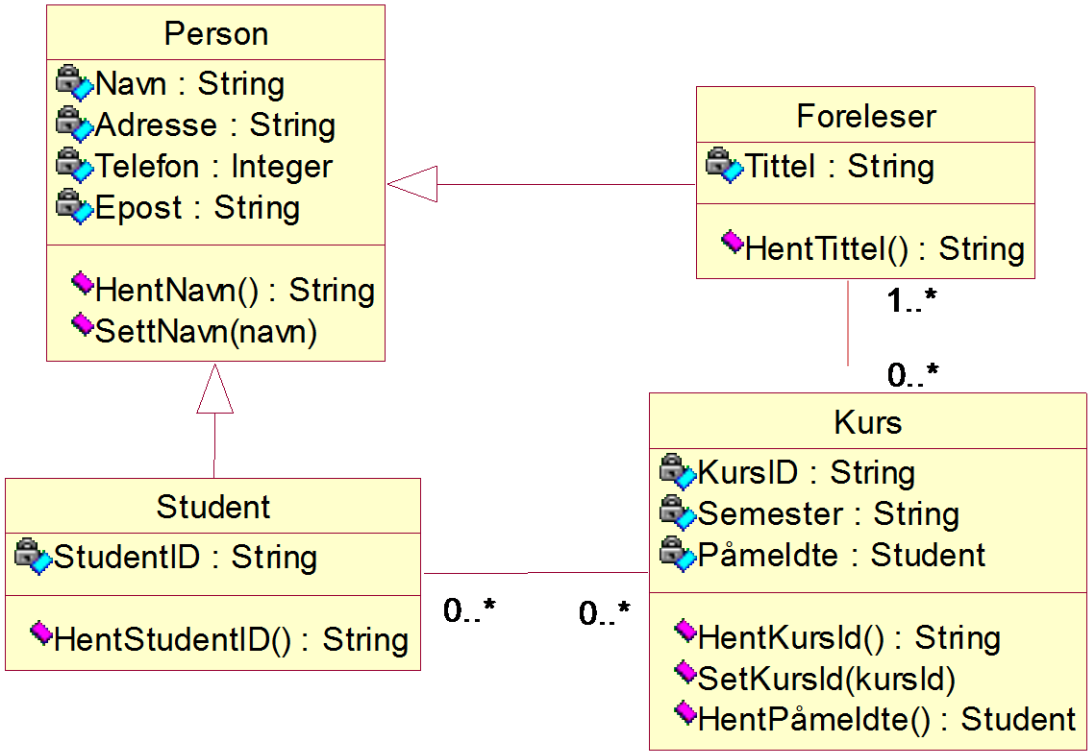
- **Mange-til-mange assosiasjon(relasjon):** En person kan ta mange kurs (0..* betyr fra 0 til mange), og et kurs kan ha mange personer (fra 0 til mange)

Klassediagram - Student tar kurs - Generalisering



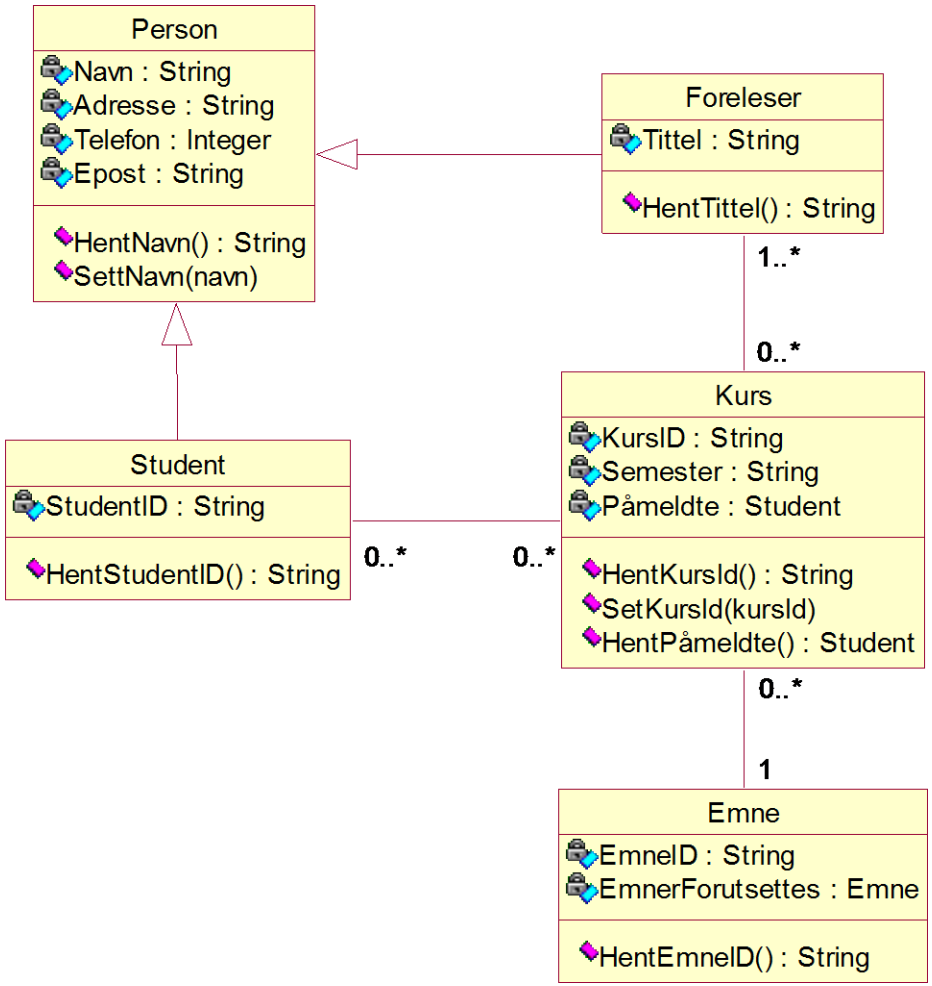
- Alle attributter og metoder i person blir arvet i Student

Klassediagram - Student tar kurs – med foreleser(e)



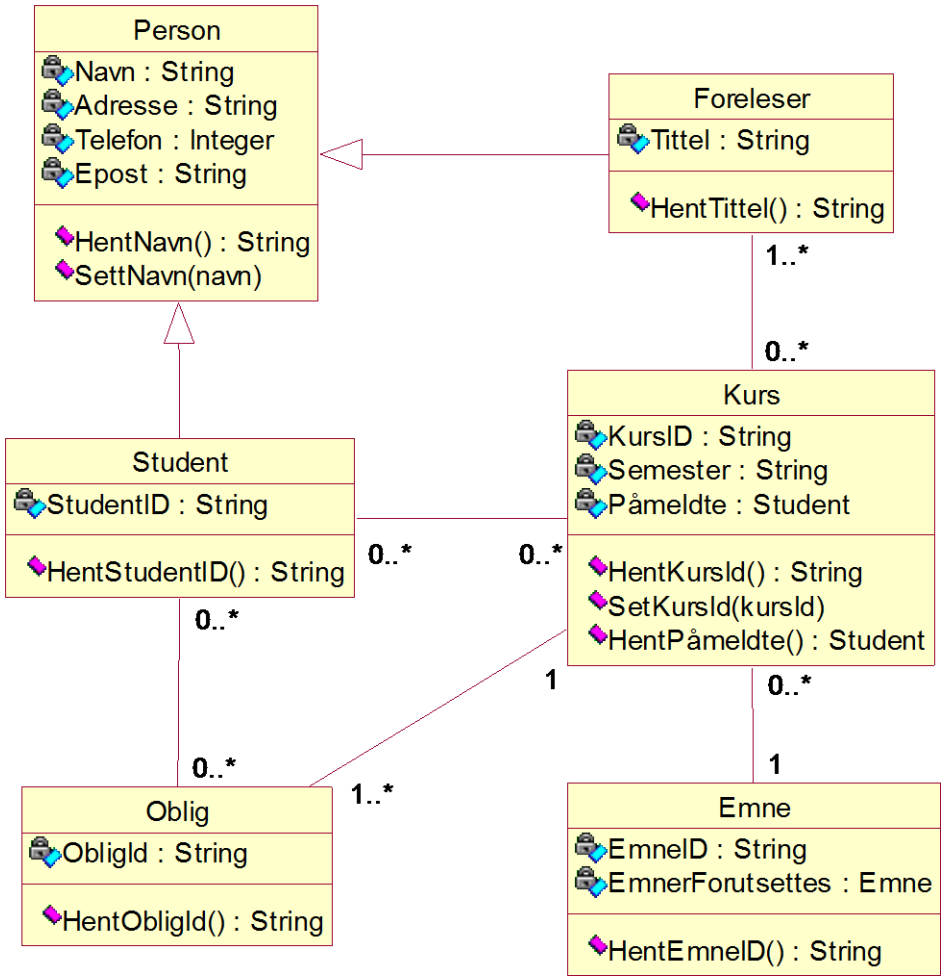
- Alle attributter og metoder i Person blir arvet i Student og Foreleser

Klassediagram - Kurs inngår i Emne



- Emne er for eksempel Systemutvikling og Kurs er for eksempel Systemutvikling-H2014

Student tar kurs – med obligatoriske oppgaver



- Alle kurs har minst en obligatorisk oppgave. Merk at her er oblig knyttet til kurs og ikke emne slik at oblig vil kunne endres neste gang kurset går.

Identifisere objektklasser

- Å identifisere objektklassene er ofte den vanskeligste delen av objektorientert design
- Det finnes ingen “magisk formel” for å identifisere objektklasser. Avhenger av kompetanse, erfaring og domenekunnskap (kunnskap om systemet og omgivelser) til utvikleren eller systemdesigneren
- Dette er en iterativ prosess. Umulig å få det riktig første gang

En tilnærming for å identifisere objektklasser

- Bruk en grammatisk tilnærming basert på en naturlig språkbeskrivelse av systemet – objekter og attributter er subjekter, operasjoner (metoder) er verb
- Bruk gjenkjennelige ting (entiteter) i applikasjonsdomenet – som Emne og Kurs, roller som Student og Foreleser
- Bruk en scenario-basert analyse og identifiser objektene, attributtene og metodene i hver scenario

Spesifikasjon av grensesnitt

- Objekt grensesnitt bør spesifiseres slik at objektene og andre komponenter kan designes i parallell.
- Ikke design representasjonen av data – kun “navn” og metoder (uten innhold). Innholdet defineres i objektene som “implementerer “grensesnittet.
- Objekter kan ha flere grensesnitt med ulike perspektiver av metodene som er spesifisert.
- Klassediagrammer blir brukt i UML for spesifikasjon av grensesnitt.

Eksempel
Ulike klasser samme grensesnitt (Interface)

```
interface Skattbar{                    // Skatt på importerte varer
        int skatt();
}

class Bil implements Skattbar{  // Bil: 100% skatt
        private String regNr;
        private int importpris;
        Bil (String reg, int imppris) {
                regNr = reg; importpris = imppris;
        }
        public int skatt( ){return importpris;}
        public String hentRegNr( ) {return regNr;}
}

class Ost  implements Skattbar{  // Ost: 200% skatt
        private int importprisPrKg;
        private int antKg;
        Ost (int kgPris, int mengde) {
                importprisPrKg = antKg; antKg = mengde;
        }
        public int skatt( ){return importprisPrKg*antKg*2;}
}
```

Finne ansvarsområder til klasser

- Hvert funksjonelle krav må tilordnes en eller flere klasser
  - Hvis en klasse har for mange ansvarsområder, vurder *å splitte* den i ulike klasser.
  - Hvis en klasse ikke har noe ansvar, så er den antageligvis *overflødig*
  - Når et ansvar ikke kan tilordnes til en eksisterende klasse, opprett en ny *klasse*.
- For å finne ansvarsområder
  - Analyser use casene
  - Se etter verb og substantiver som beskriver *handlinger*

Objektdesign: Ansvarstilordning

- Ansvar er knyttet til objektet i form av dets oppførsel
  - *Handling:* Opprette objekt, beregne, initiere handlinger i andre objekter, kontrollere og koordinere handlinger i andre objekter.
  - *Kunnskap:* Vite om private data, relaterte objekter, ting som det kan utlede eller beregne
- Ansvar er ikke det samme som metoder, men metoder implementeres for å oppfylle ansvaret
- Kategorier av ansvar:
  - Sette (set) og hente (get) verdier av attributter
  - Opprette og initialisere nye instanser (objekter)
  - Hente fra og lagre til persistent minne (database)
  - Slette instanser
  - Legge til og slette linker for assosiasjoner
  - Kopiere, konvertere og endre
  - Beregne numeriske resultater
  - Navigere og søke
  - …

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 implementasjonsdetaljer fra andre objekter

- Grady Booch

Modularisering

- Høy kohesjon
  - Et objekt skal bare ha ansvar for relaterte ting
- Lav kobling
  - Et objekt skal samarbeide 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 oppgaver 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

Designmodellen

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

Analyse- vs. designmodell

- *Analysemodellen* utelater mange klasser som er nødvendige i et komplett system
  - Er typisk en domenemodell
  - Kan inneholde mindre enn halvparten av klassene i systemet.
  - Uavhengig av spesielle
    - brukergrensesnittsklasser
    - arkitekturklasser (f.eks. design patterns klasser)
- Den komplette *designmodellen* inneholder
  - Domenemodellen
  - Brukergrensesnittsklasser
  - Arkitekturklasser (f.eks. slik at klasser kan kommunisere)
  - Utility klasser (f.eks. håndtering av mengder og strenger)

Objektorientert designprosess

- I en objektorientert designprosess utvikles flere ulike systemmodeller
- Det kreves mye for å utvikle og vedlikeholde disse modellene, og for små systemer er det ofte ikke kostnadseffektivt
- For store systemer, som utvikles av ulike grupper og team, vil slike designmodeller være en viktig mekanisme for kommunikasjon

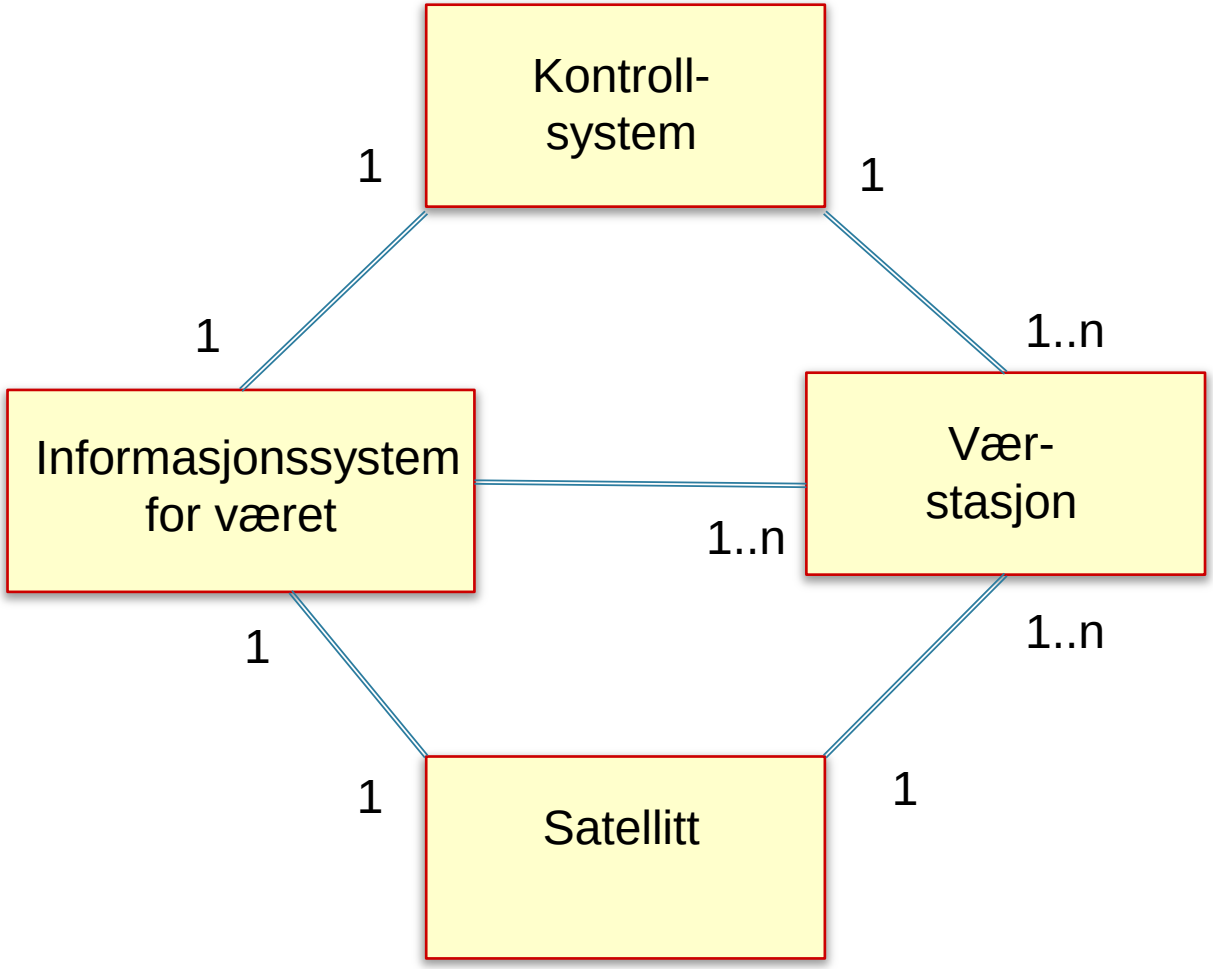
Designprosesser

- Det finnes flere ulike objektorienterte designprosesser avhengig av organisasjonen som bruker prosessen
- Felles aktiviteter i disse prosessene inkluderer
  - Forstå og definere systemets kontekst og eksterne interaksjoner med systemet
  - Designe systemarkitekturen
  - Identifisere nøkkelobjektene i systemet
  - Utvikle designmodeller
  - Spesifisere grensesnitt
- UML aktivitetsdiagram brukes til å definere forretningsprosessmodeller

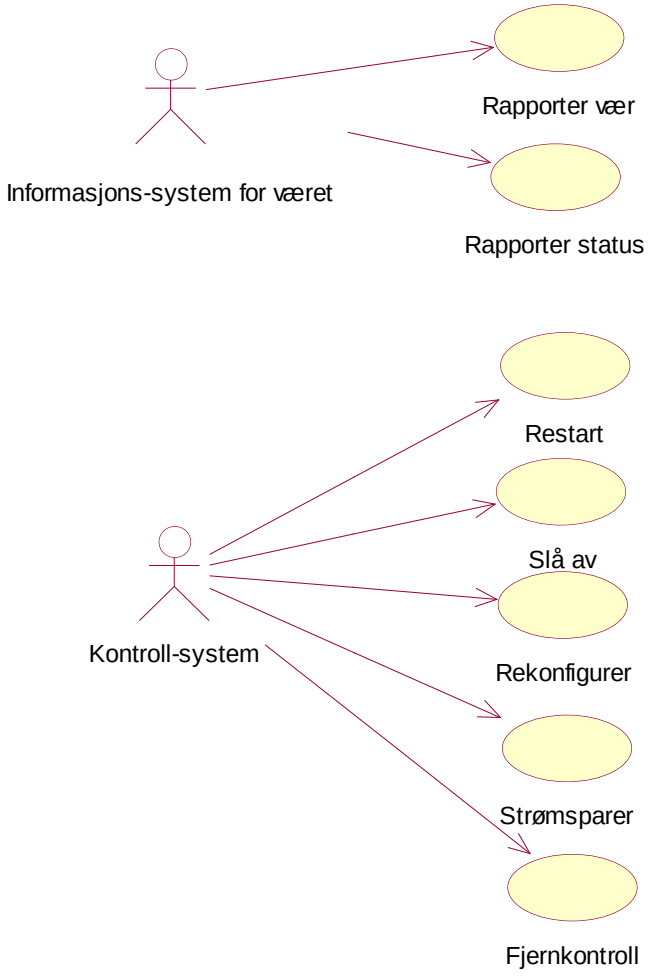
Systemkontekst og interaksjoner

- Å forstå sammenhengen mellom programvaren som skal utvikles og ytre omgivelser er viktig for hvordan man beskriver systemfunksjonalitet og hvordan man strukturerer systemet for å kommunisere med dets omgivelser
- Å forstå konteksten til systemet vil hjelpe til å definere hvilken funksjonalitet som defineres i systemet og hva som er i de assosierte systemer.

Systemkontekst for værstasjon



Værstasjon – Use cases



Use case beskrivelse: Rapporter vær

System	Værstasjon
Use case	Rapporter vær
Aktører	Informasjonssystem for vær, Værstasjon
Beskrivelse	Værstasjonen sender et sammendrag av værdata, som har blitt samlet fra instrumentene i innsamlingsperioden, til informasjonssystemet. Dataene som sendes er maks, min og gjennomsnittstemperatur; maks, min og gjennomsnitt lutftrykk; maks, min og gjennomsnitt vindhastighet, total nedbør (i mm), og vindretninger for hvert 5 minutt.
Stimulu	Informasjonssystemet etablerer en satelitt kommunikasjonslink med værstasjonen og spør om overføring av data.
Respons	Sammendrag av data sendes til informasjonssystemet
Kommentarer	Værstasjoner rapporterer vanligvis hver time, men dette varierer litt fra stasjon til stasjon og vil kunne endres i framtiden

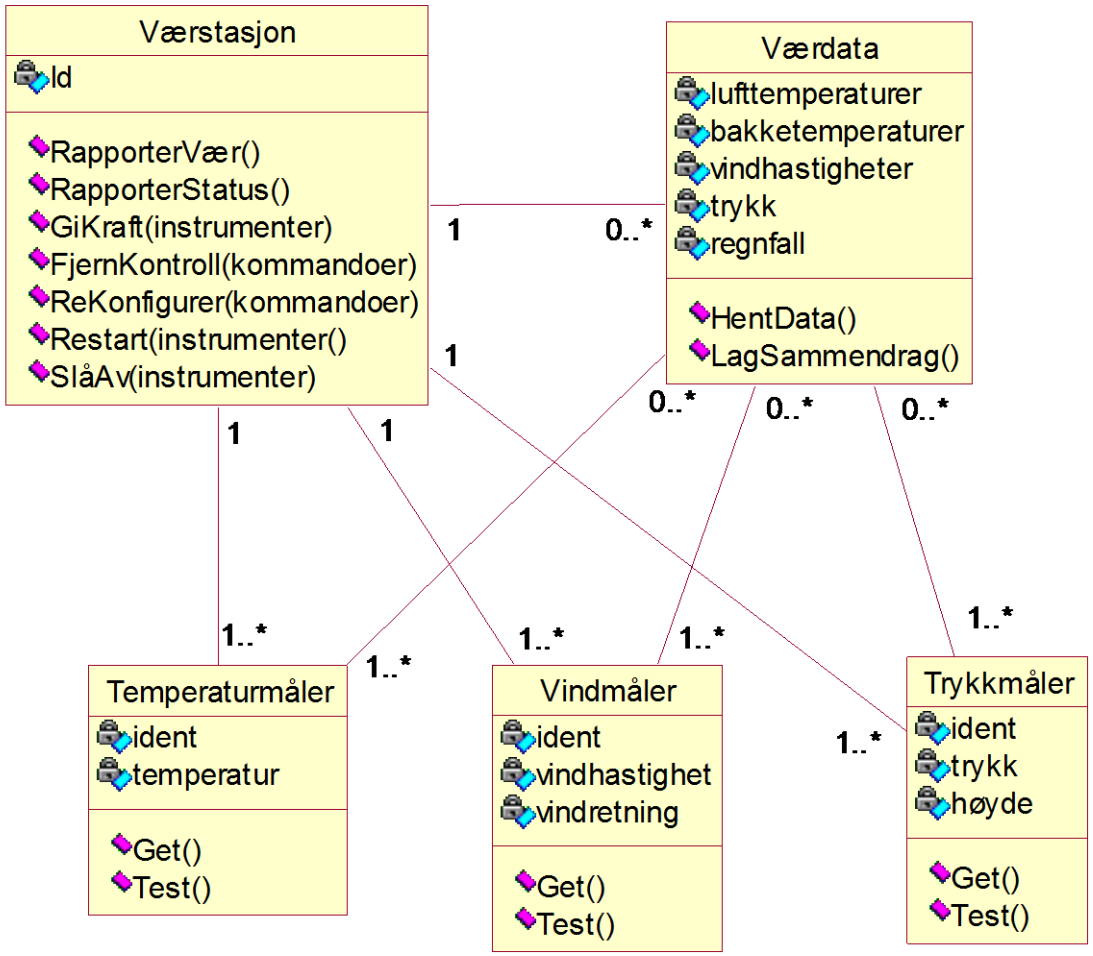
Beskrivelse av værstasjon

- En værstasjon er en pakke instrumenter, kontrollert av programvare, som samler data, gjør dataprosessering og overfører dataene for videre prosessering. Instrumentene inkluderer luft- og bakketermometer, en vindmåler, et barometer og ett instrument som tar imot og måler nedbørsmengde. Data blir samlet periodevis (for eksempel hver time)
- Ved overføring av værdata, vil værstasjonen prosessere og sammenfatte innsamlet data. Dataene blir overført til en datamaskin der forespørselen blir mottatt.

Objektklasser for værstasjon

- Basert på håndgripelige “hardware” og data i systemet
  - Bakketermometer, vindmåler, barometer
    - Applikasjons domeneobjekter som er relatert til instrumenter i systemet
  - Værstasjon
    - Basis grensesnitt fra værstasjon til omgivelsene. Værstasjonen reflekterer interaksjoner som identifiseres i use-case modellen
  - Værdata
    - Samler værdataene som kommer fra instrumentene

Objektklasser for Værstasjon – med assosiasjoner



Designmønstre – ”Design patterns”

- Et designmønster er en måte å gjenbruke abstrakt kunnskap om et problem og løsningen på problemet
- Et mønster er en beskrivelse av et problem og essensen av løsningen
- Bør være tilstrekkelig abstrakt til å kunne bli gjenbrukt i ulike situasjoner
- I beskrivelser av mønstre brukes som regel objektorienterte teknikker som arv og polymorfisme (”Virtual methods”)

Hvorfor bruke mønstre (patterns) for informasjonssystemer?

- Mye av vårt daglige virke består i å lete etter strukturer (mønstre) i omgivelsene
- Mønstre er vanlige løsninger på vanlige problemer
- Det samme gjelder utvikling av informasjonssystemer
- Mange systemer har grunnleggende fellestrekk
- Det finnes mange mønstre (patterns) som er blitt utviklet gjennom systemutviklingens historie

Mer om Mønstre (”patterns”)

- Mønstre er navngitte retningslinjer for hvordan ansvar skal fordeles i ulike situasjoner.
- Mønstre brukes bl.a. i prosessen med å forfine sekvensdiagrammer
- GRASP – ’Patterns of General Principles in Assigning Responsibilites’ = Mønster for problem/løsning
- Sentrale mønstre er
  - Ekspertprinsippet:
    - La det objektet som har kunnskapen (dataene) også behandle den
  - Kontrollobjektprinsippet:
    - To typer kontrollere:
      - Fasadekontroller: En kontrollklasse har ansvar for alt (brukes i et lite system)
      - Use case kontroller: Styrer ett use case (brukes i større systemer. Ett kontrollobjekt for hvert use case).
  - Skaperprinsippet:
    - Legg ansvar for å opprette et nytt objekt i klassen som må vite om det nye objektet

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:
  - Eks: Student-kurs:

    Hvilket objekt har ansvar for å vite om hvilke emner som kreves for å ta et gitt emne?

    Hvilket objekt har ansvar for å gi en liste over alle studentene på et kurs?

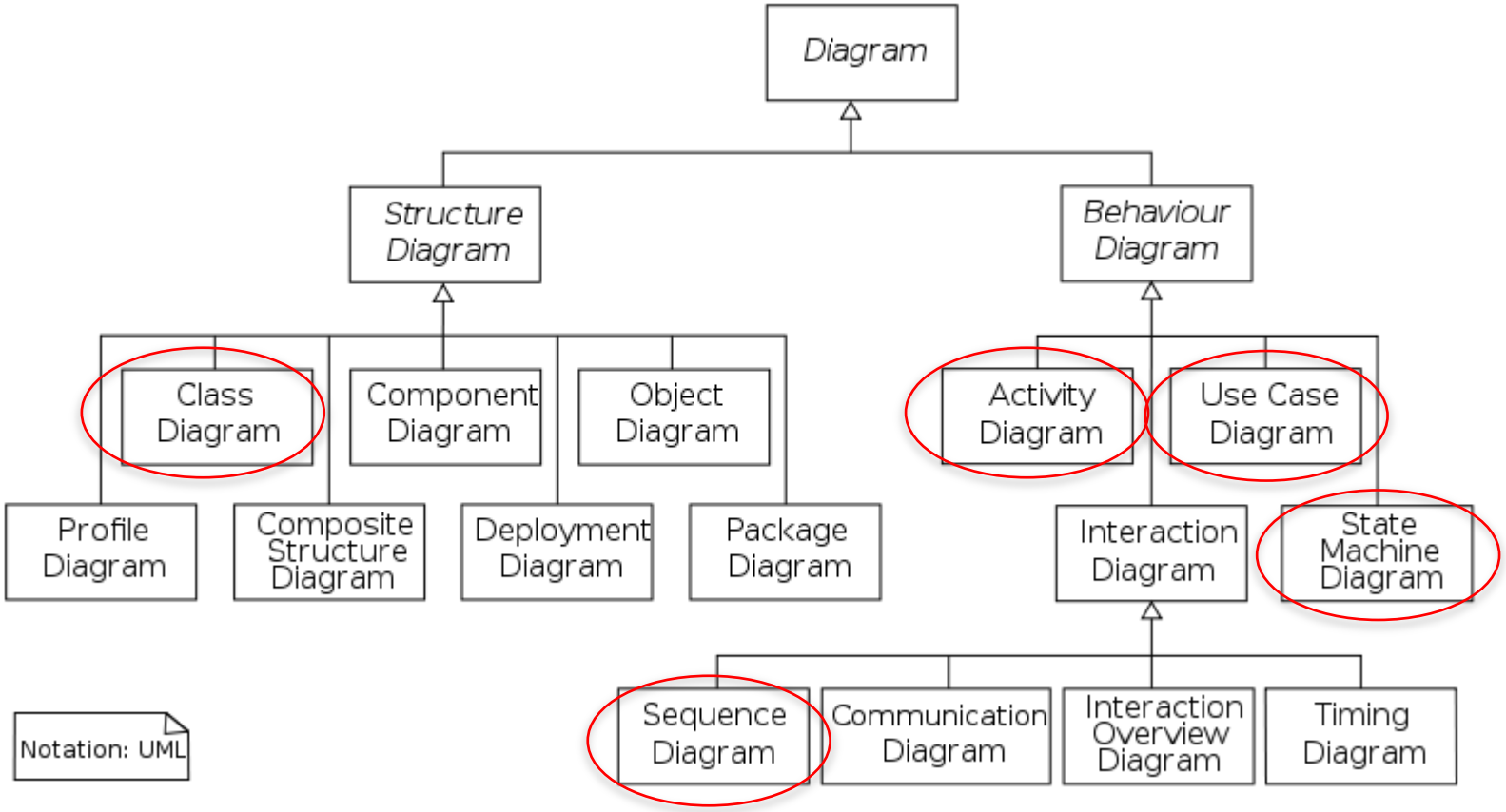
Skaperprinsippet (Creator)

- 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

Kontrollobjektprinsippet (Controller)

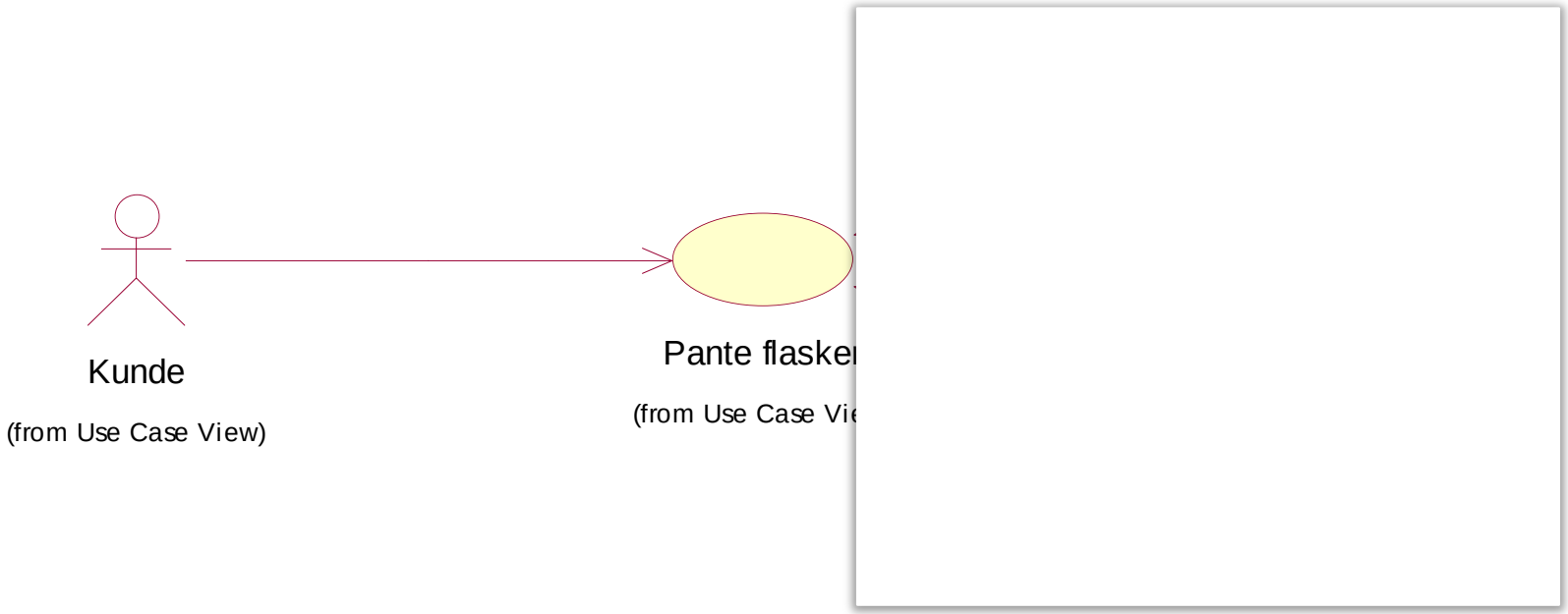
- Hvilken klasse skal behandle en systemhendelse/melding?
  - Kontrolleren ligger gjerne på klienten
  - Kontrolleren har bare metoder, få eller ingen attributter
  - Kontrolleren gjør ikke jobben selv, men mottar og fordeler oppgaver – er en slags administrator
  - Delegerer oppgaver og styrer use case
  - Er et bindeledd mellom brukergrensesnittet og applikasjonslaget

UML - diagrammer



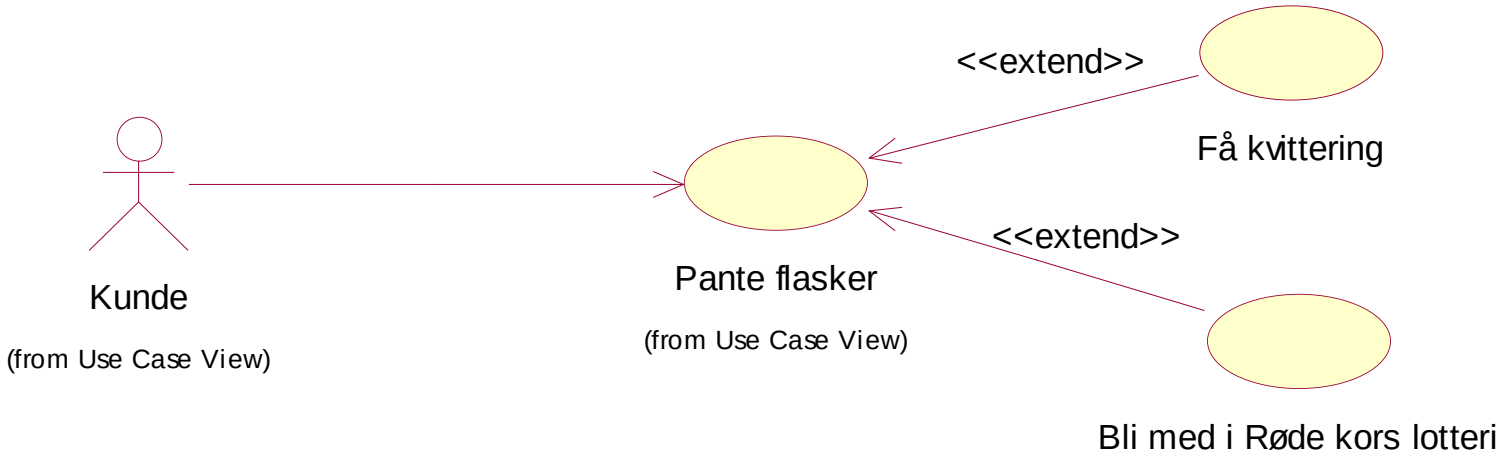
Kilde: http://en.wikipedia.org/wiki/Unified_Modeling_Language#Structure_diagrams

Use Case – Pante flasker



- Use Case diagrammer brukes til å beskrive interaksjonen mellom brukere og systemer..

Use Case – Pante flasker



Tekstlig beskrivelse for “Pante flasker”

Prebetingelse: Panteautomat er klar til å ta imot pant
Postbetingelse: Kunde får ut kvittering eller lodd i røde kors trekning

Tekstlig beskrivelse for “Pante flasker”

Prebetingelse: Panteautomat er klar til å ta imot pant
Postbetingelse: Kunde får ut kvittering eller lodd i røde kors trekning

Hovedflyt:
1. Kunde setter inn en flaske (eller et panteobjekt)
2. Panteautomaten skanner koden til flasken (panteobjektet) som ble puttet inn
3. Objektet er godkjent, pantebeløpet blir lagt til det totale beløpet
4. Kunde trykker på kvittering
5. Panteautomat skriver ut kvittering

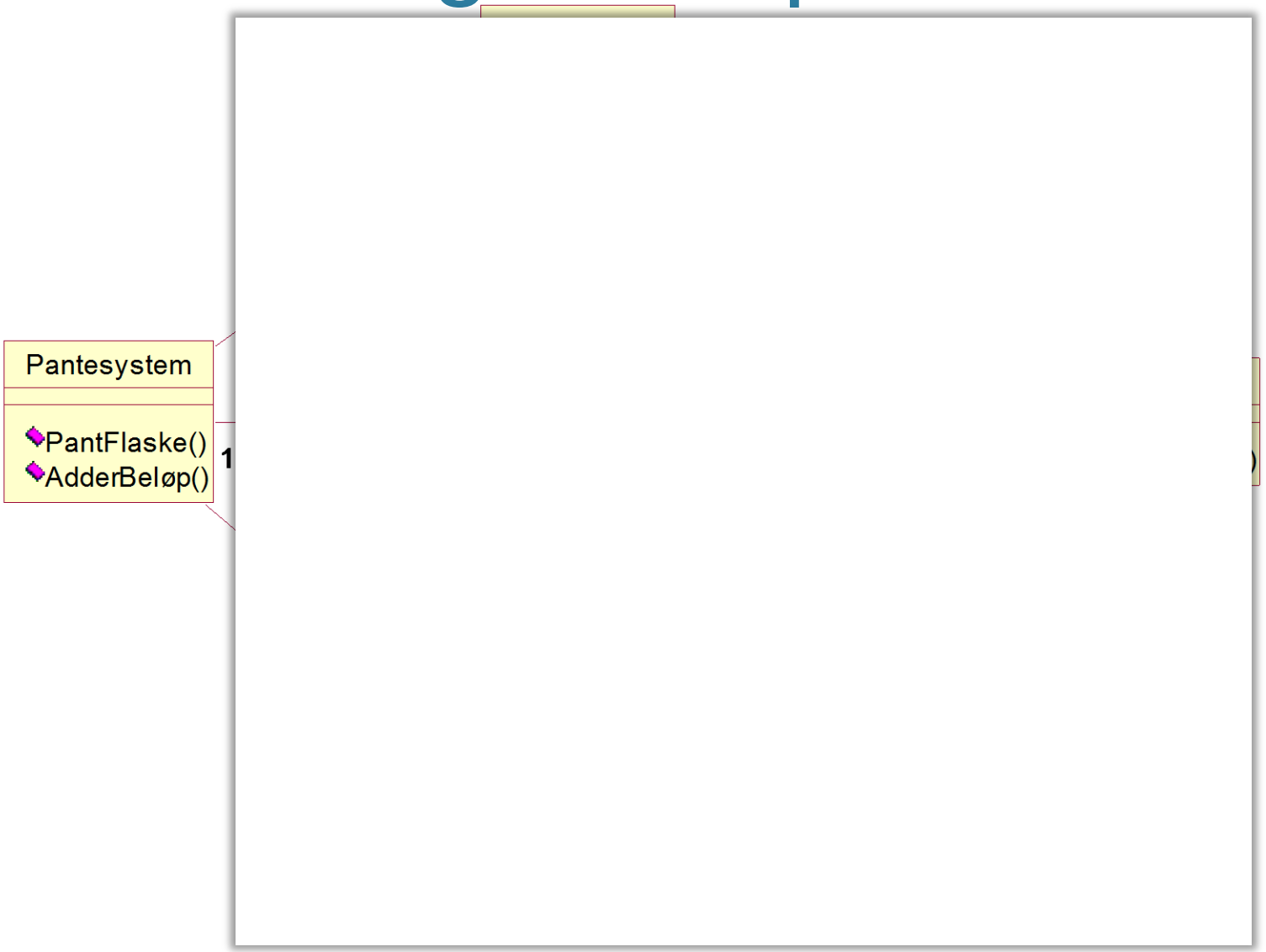
Tekstlig beskrivelse for “Pante flasker”

Prebetingelse: Panteautomat er klar til å ta imot pant
Postbetingelse: Kunde får ut kvittering eller lodd i røde kors trekning

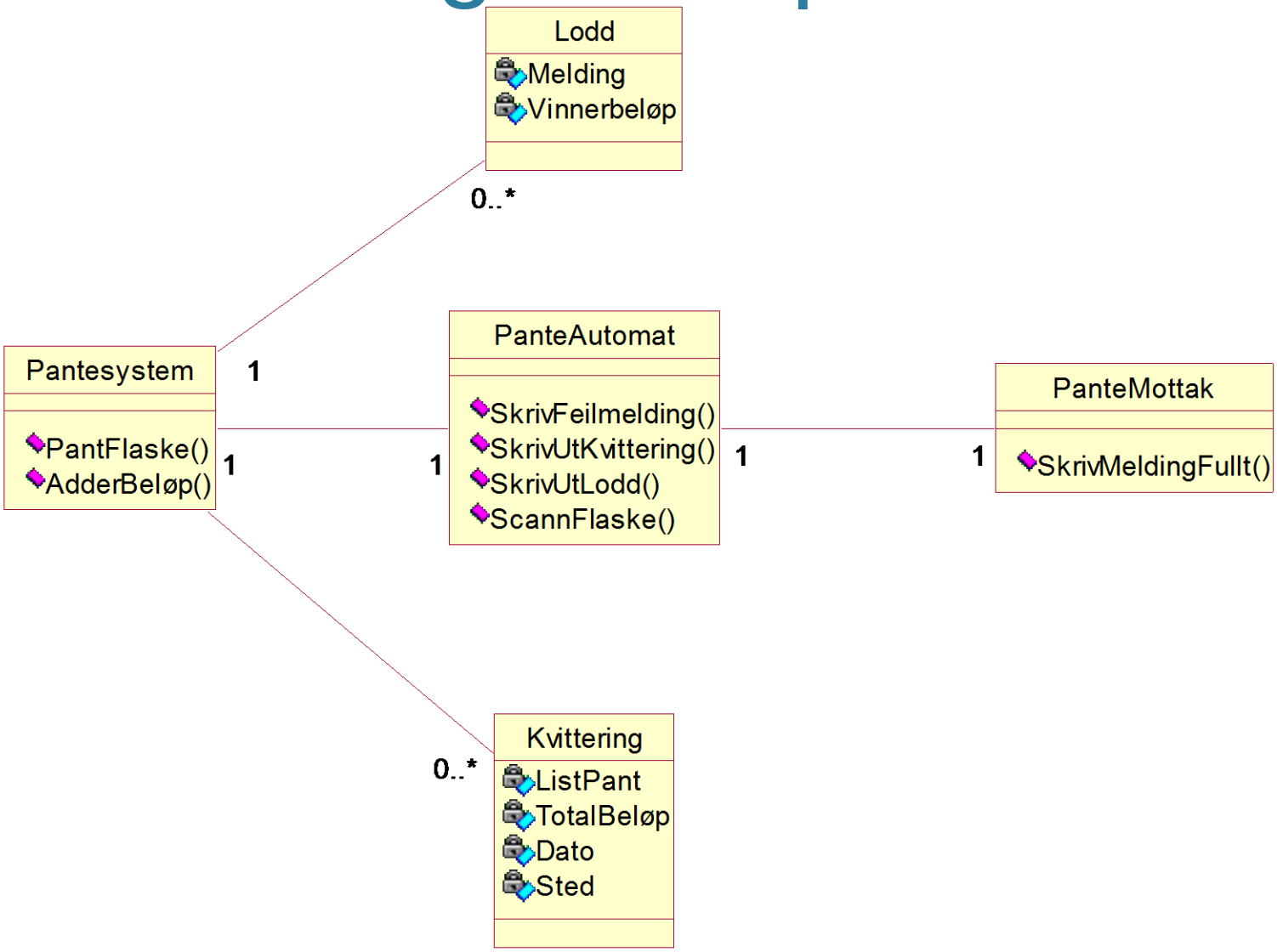
Hovedflyt:
1. Kunde setter inn en flaske (eller et panteobjekt)
2. Panteautomaten skanner koden til flasken (panteobjektet) som ble puttet inn
3. Objektet er godkjent, pantebeløpet blir lagt til det totale beløpet
4. Kunde trykker på kvittering
5. Panteautomat skriver ut kvittering

Alternative flyt
3.1 Objekt ikke godkjent
3.2 Start fra 1
4.1.A1 Kunde trykker på ”Røde kors lotteri”
4.2.A1 Kunde skriver ut Røde kors lodd
4.1.A2 Kunde setter inn ny flaske (panteobjekt)
4.2.A2 Start fra 1

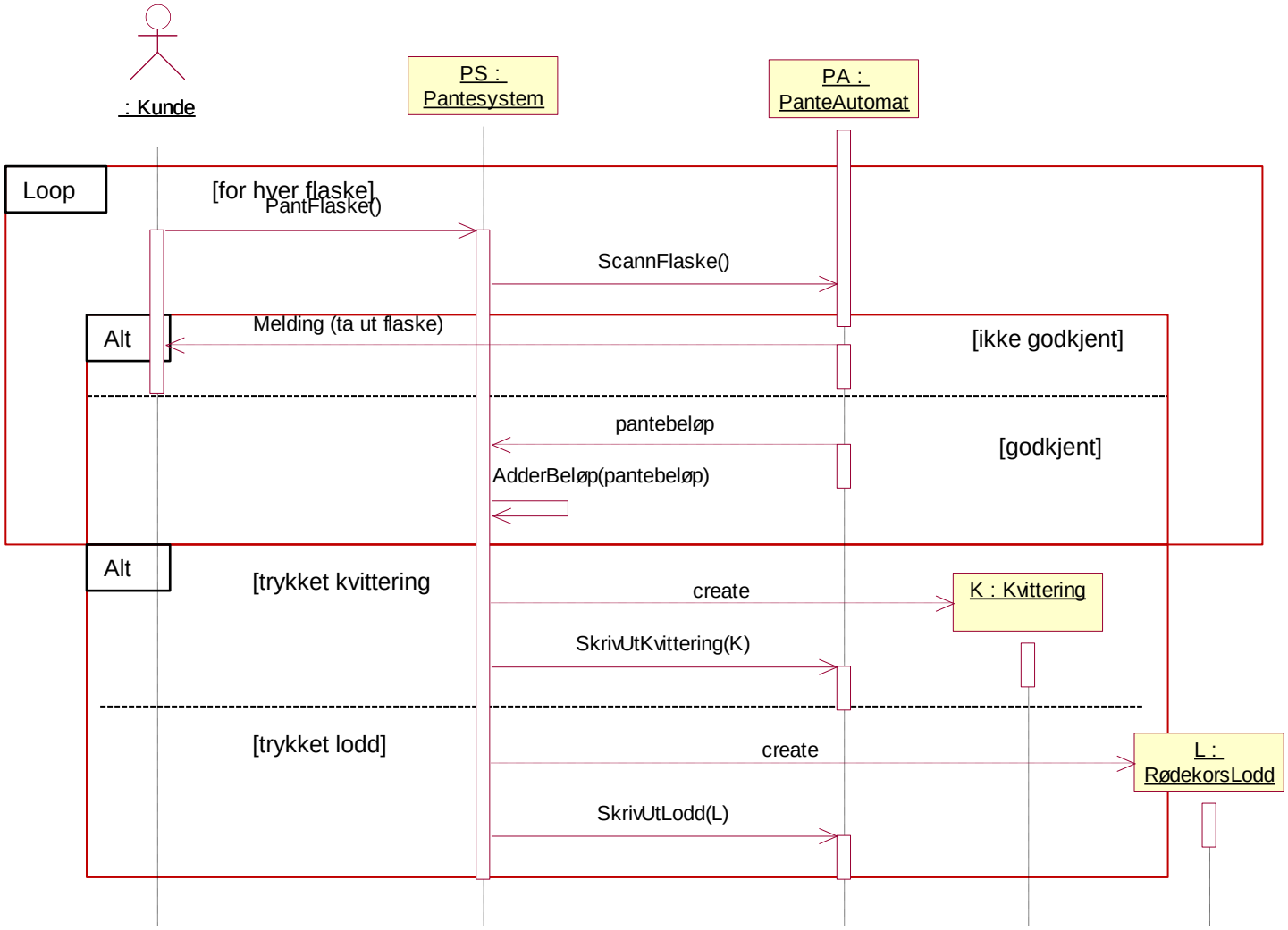
Klassediagram – pante flasker



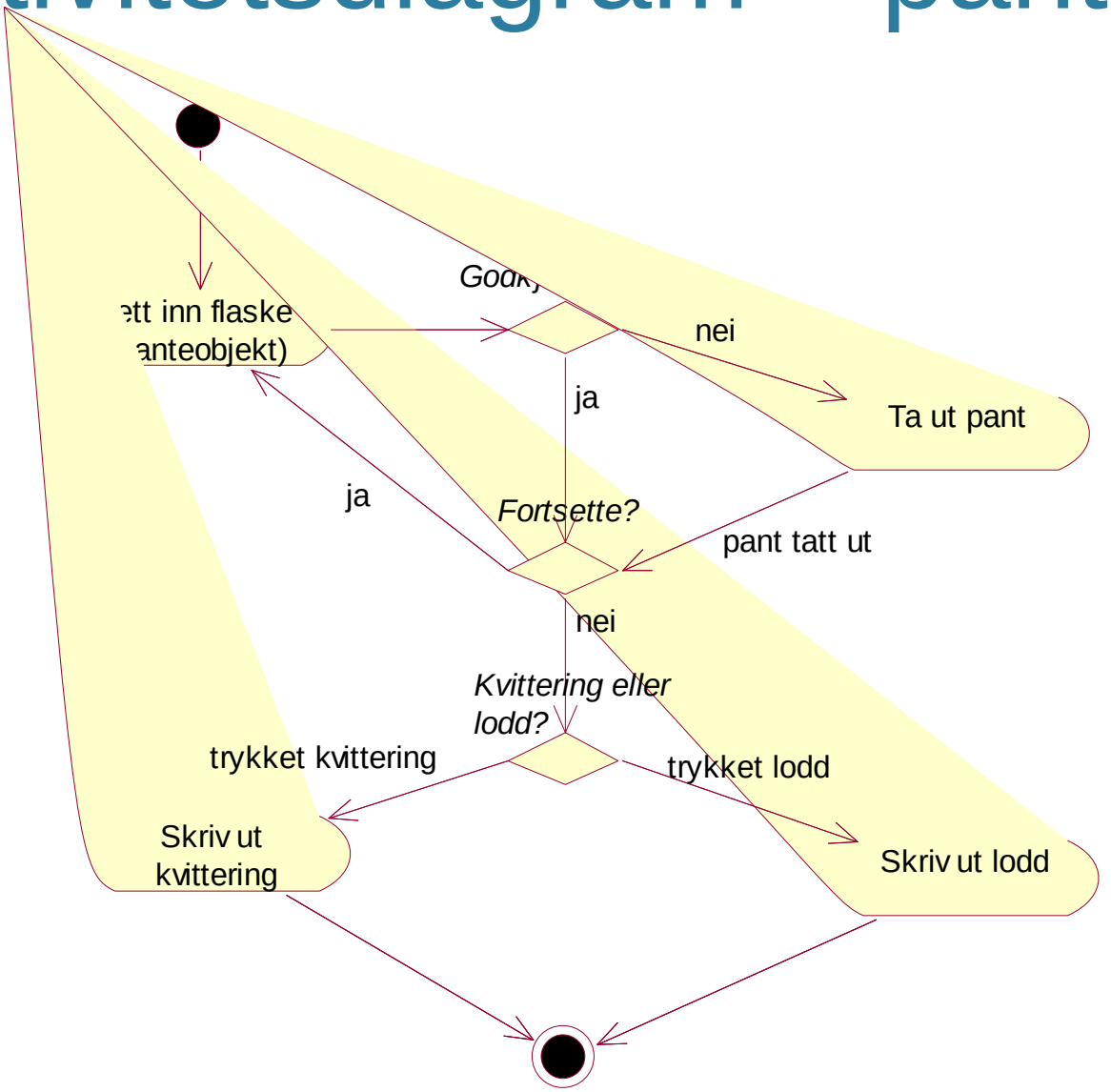
Klassediagram – pante flasker



Sekvensdiagram – pante flasker



Aktivitetsdiagram – pante flasker



Tilstandsdiagram - panteautomat