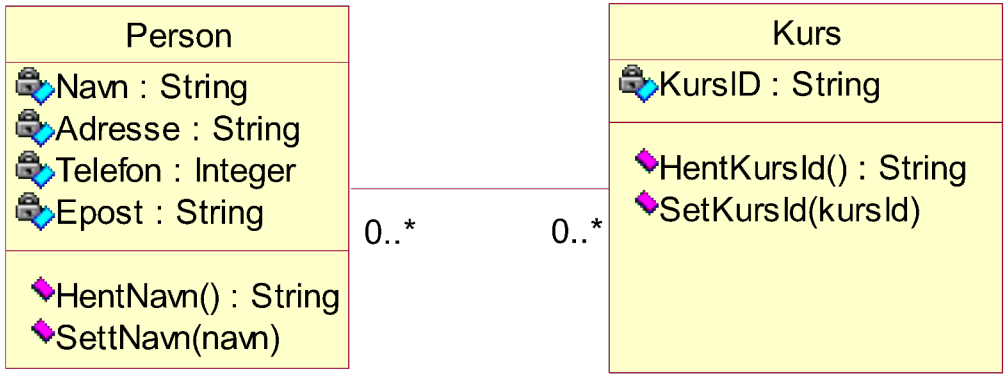
IN1030: 05. april 2022

Mer om objektorientering og UML

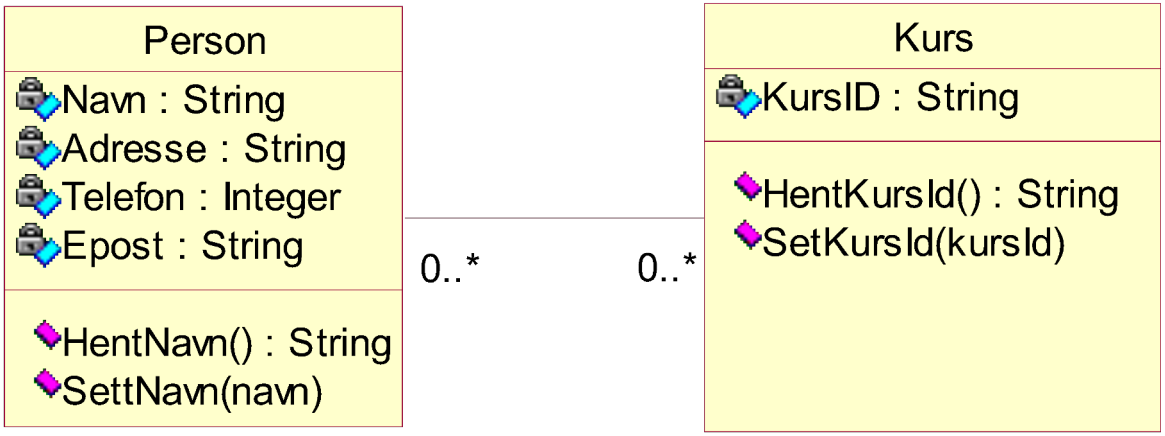


Yngve Lindsjørn

ynglin@ifi.uio.no

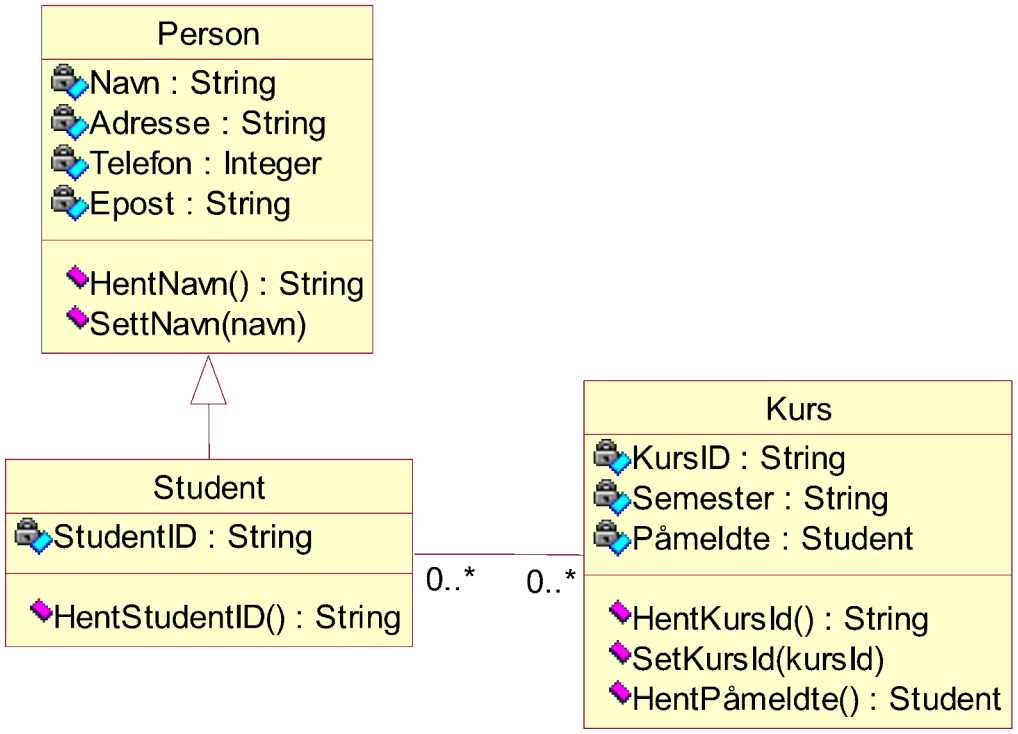
Temaer i dagens forelesning

- Klassediagram
- Objektorientert design
- Designmodeller
- Designmønstre (”Design Patterns”)
- Eksempler på diagrammer- bruk av UML
- Eksempel på kode (pseudo kode) fra sekvensdiagram



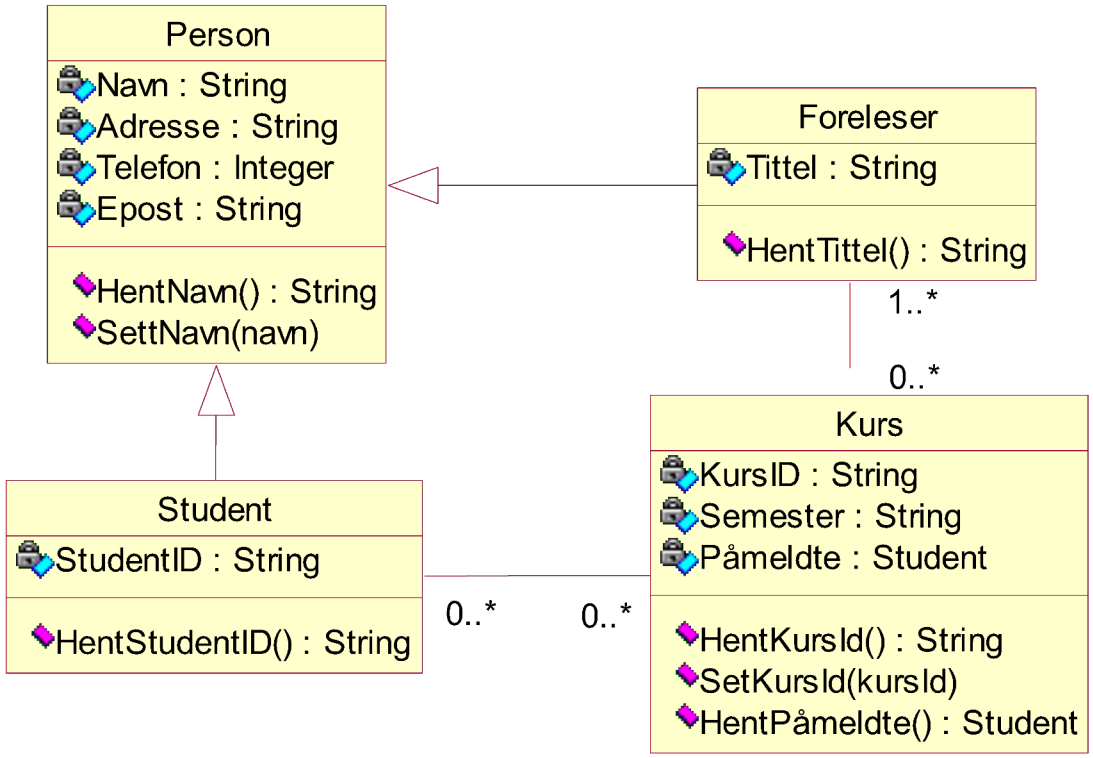
- **Mange-til-mange assosiasjon(relasjon):** En person kan ta mange kurs (0..* betyr fra 0 til mange), og et kurs kan ha mange personer (fra 0 til mange)

Klassediagram - Student tar kurs - Generalisering



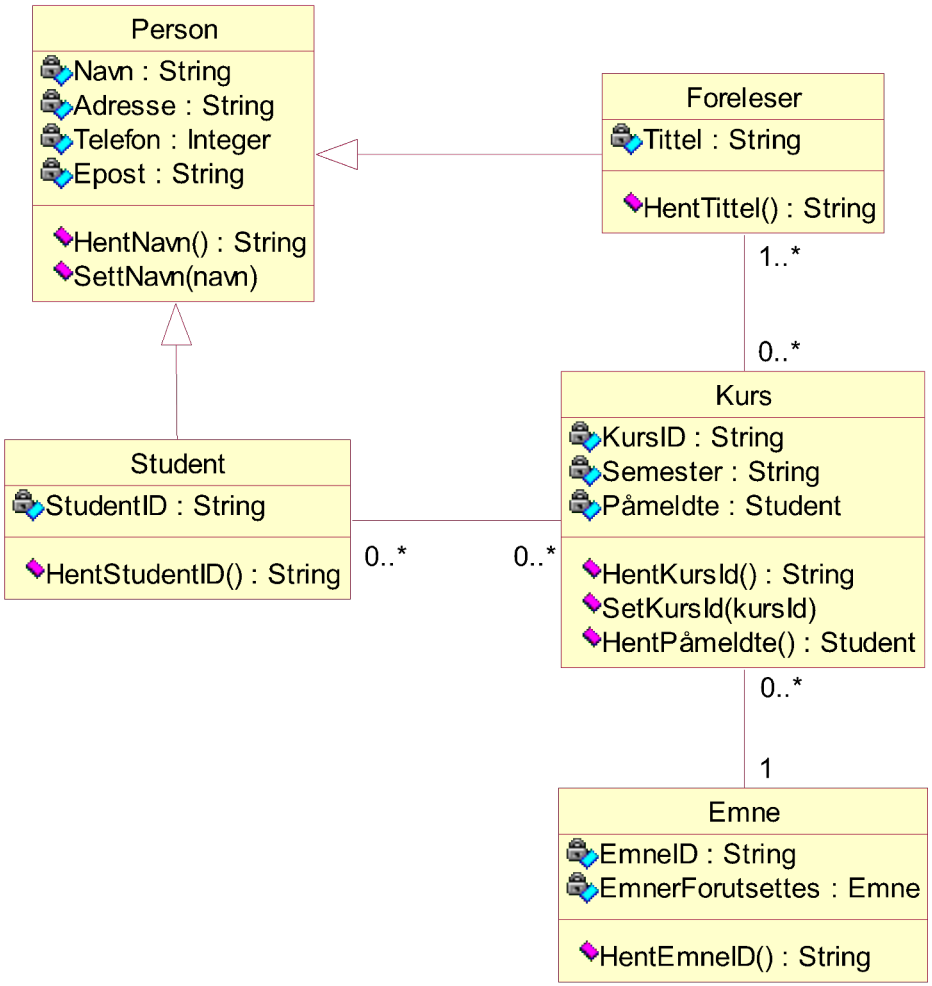
- Alle attributter og metoder i person blir arvet i Student

Klassediagram - Student tar kurs – med foreleser(e)



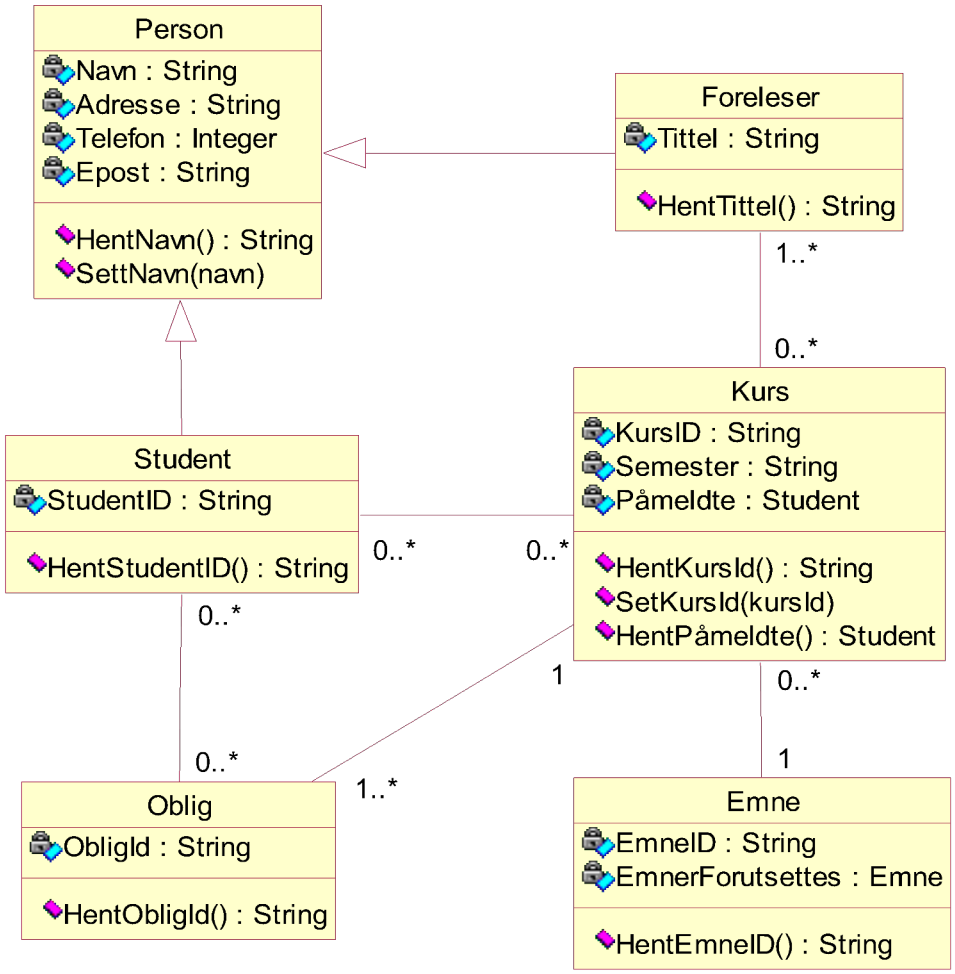
- Alle attributter og metoder i Person blir arvet i Student og Foreleser

Klassediagram - Kurs inngår i Emne



- Emne er for eksempel IN1030 og Kurs er for eksempel IN1030V20

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 klasser

- Det finnes ingen “magisk formel” for å identifisere klasser.
  - Avhenger av kompetanse, erfaring og domenekunnskap (kunnskap om systemet og omgivelser) til utvikleren eller systemdesigneren
- Iterativ prosess. Umulig å få det riktig første gang

En tilnærming for å identifisere klasser

- Bruk en grammatisk tilnærming – objekter og attributter er subjekter, operasjoner (metoder) er verb
- Bruk gjenkjennelige ting (entiteter) – 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/interface

- Grensesnitt/Interface 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 (brukt i IN1010)
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
- Metoder brukes for å oppfylle ansvaret
- Kategorier av ansvar:
  - Sette og hente verdier av attributter
  - Opprette 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 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 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 objekter kan kommunisere)
  - Utility klasser (f.eks. håndtering av mengder og strenger)

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

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
- Sentrale prinsipper 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 ta ansvaret
- **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?

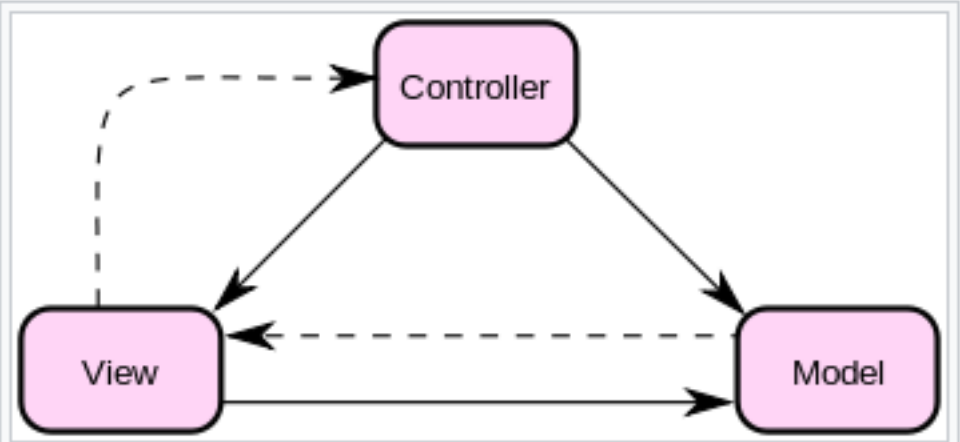
Skaperprinsippet (Creator)

- 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)

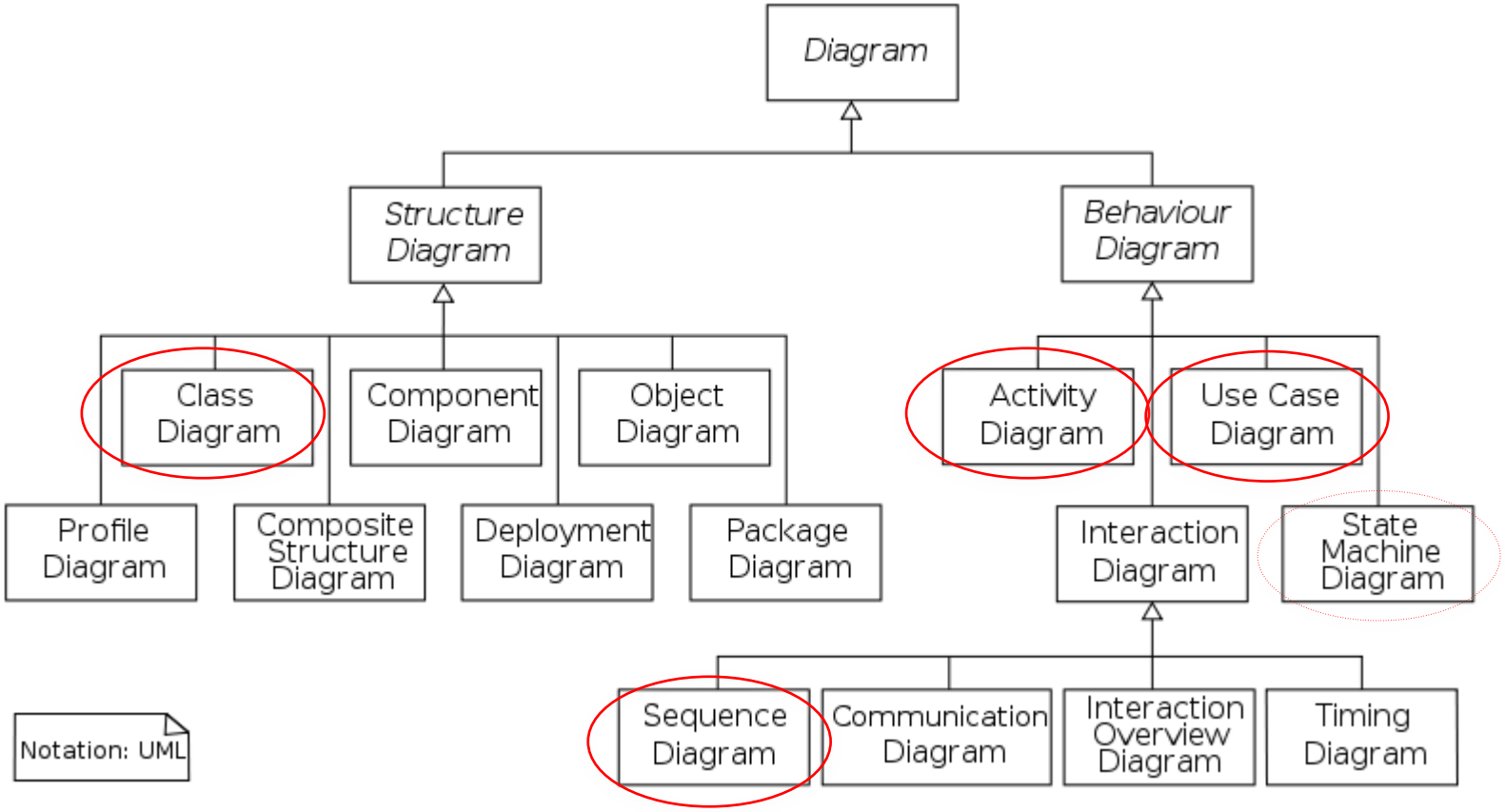
- Hvilken klasse skal behandle en hendelse/melding?
  - Kontrolleren ligger gjerne på klienten
  - Kontrolleren har bare metoder, få eller ingen attributter
  - Kontrolleren gjør ikke jobben selv, men mottar og fordeler oppgaver – er en slags administrator
  - Delegerer oppgaver og styrer use case
 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

MVC – Model View Controller



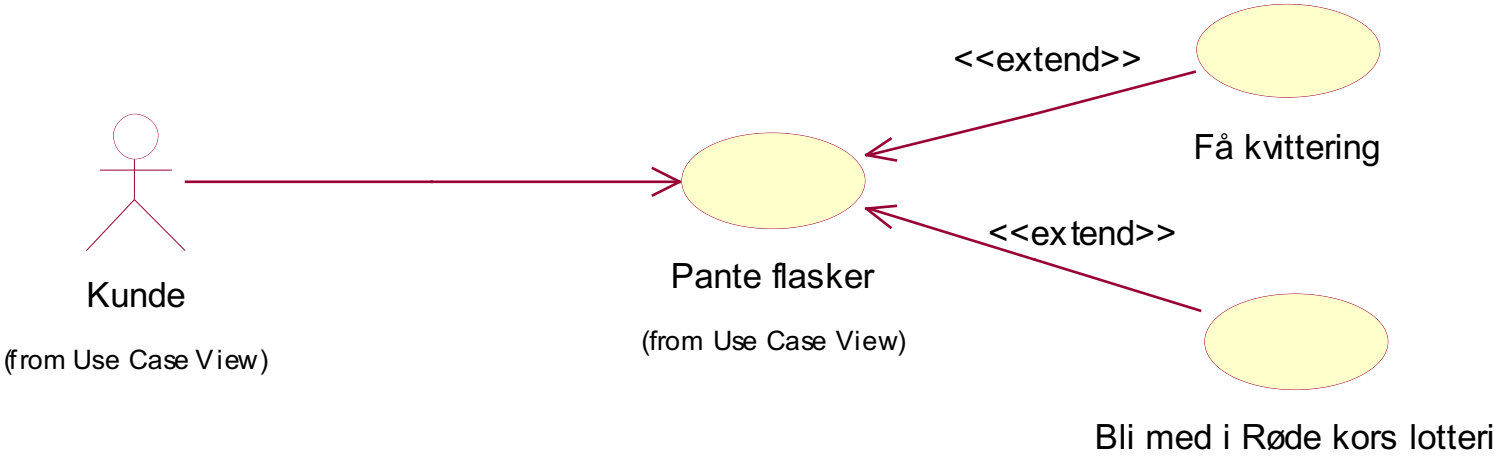
- **Model**: *Holder på data* (eks. personobjekt som er en instans av en klasse med definerte med egenskaper og metoder)
- **View**: *Viser fram data*
- **Controller**: *Flytter på data* - kommuniserer mellom View og Model (f.eks. ved muse eller tastatur trykk)

UML - diagrammer



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

Use Case – Pante flasker



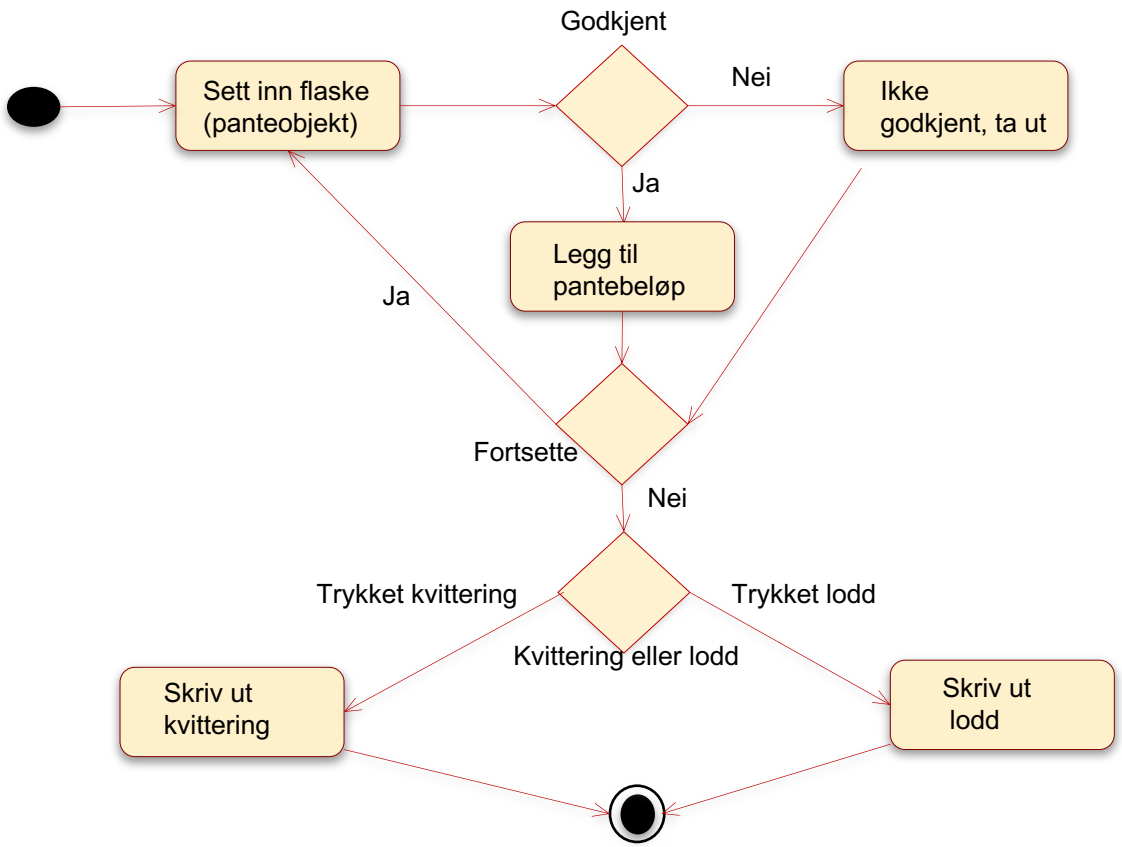
Tekstlig beskrivelse for “Pante flasker”

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

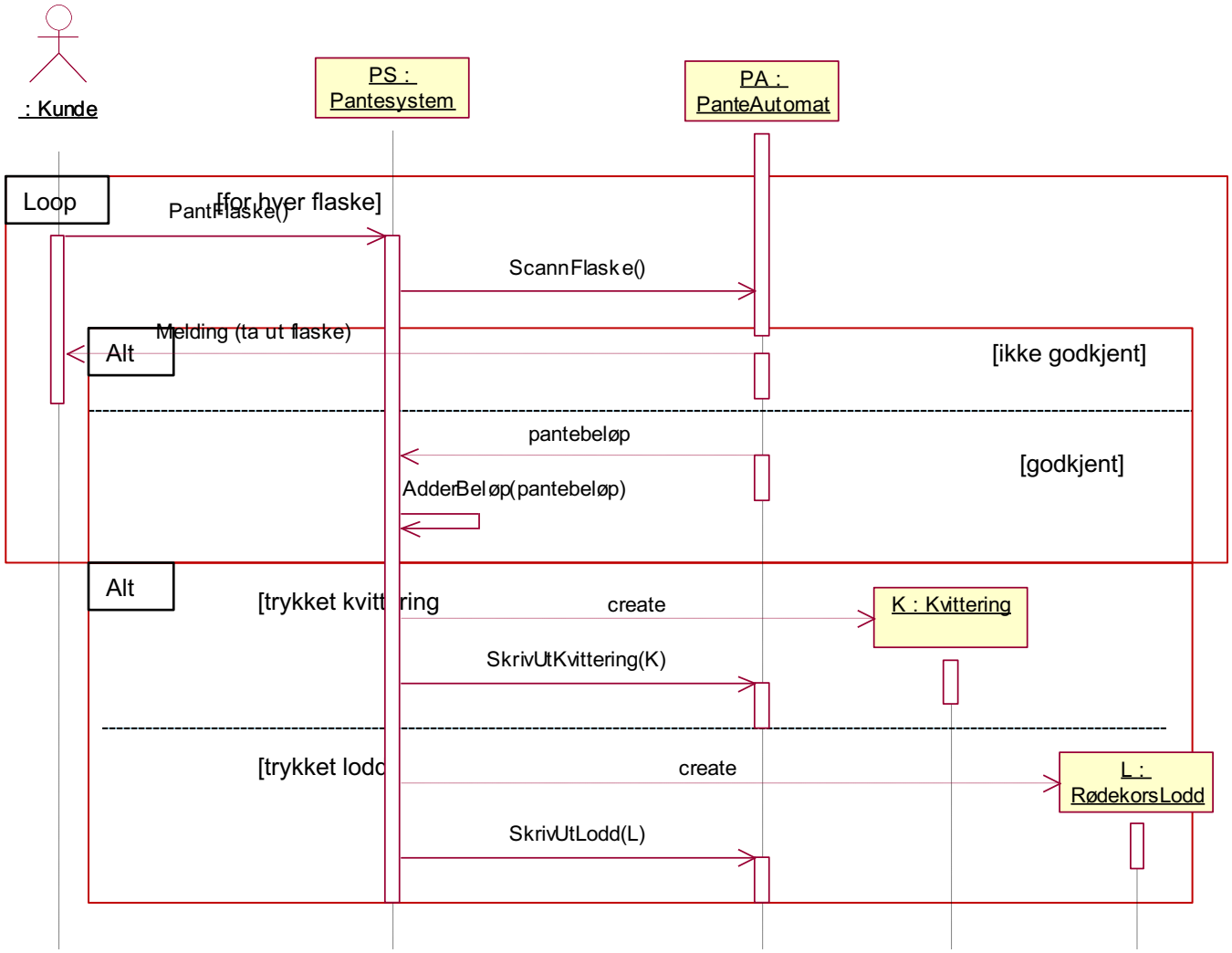
Hovedflyt:
1. Kunde setter inn en flaske (eller et panteobjekt)
2. Panteautomaten skanner koden til flasken (panteobjektet) som ble puttet inn
3. Objektet er godkjent, pantebeløpet blir lagt til det totale beløpet
4. Kunde trykker på kvittering
5. Panteautomat skriver ut kvittering

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

Aktivitetsdiagram – pante flasker

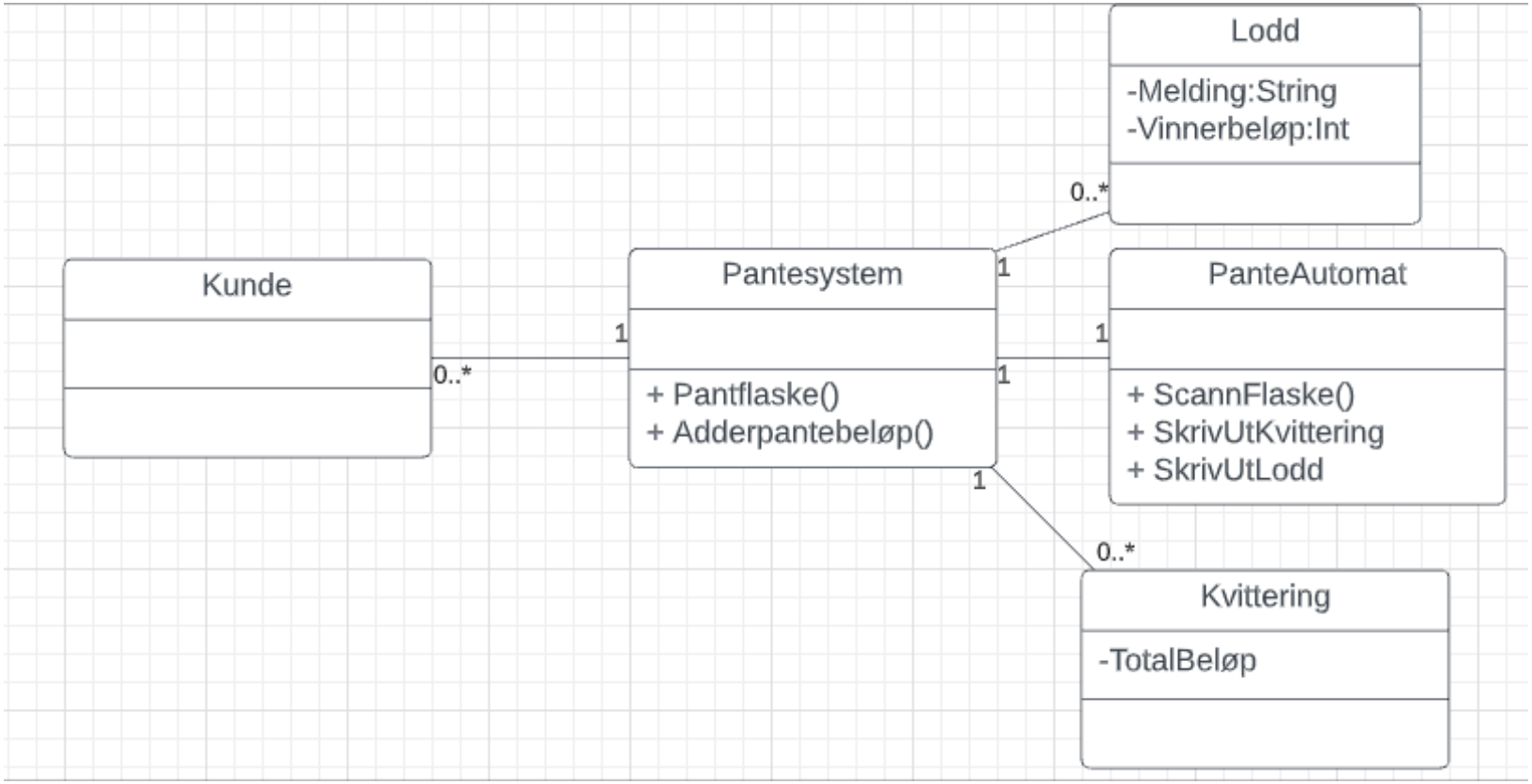


Sekvensdiagram – pante flasker

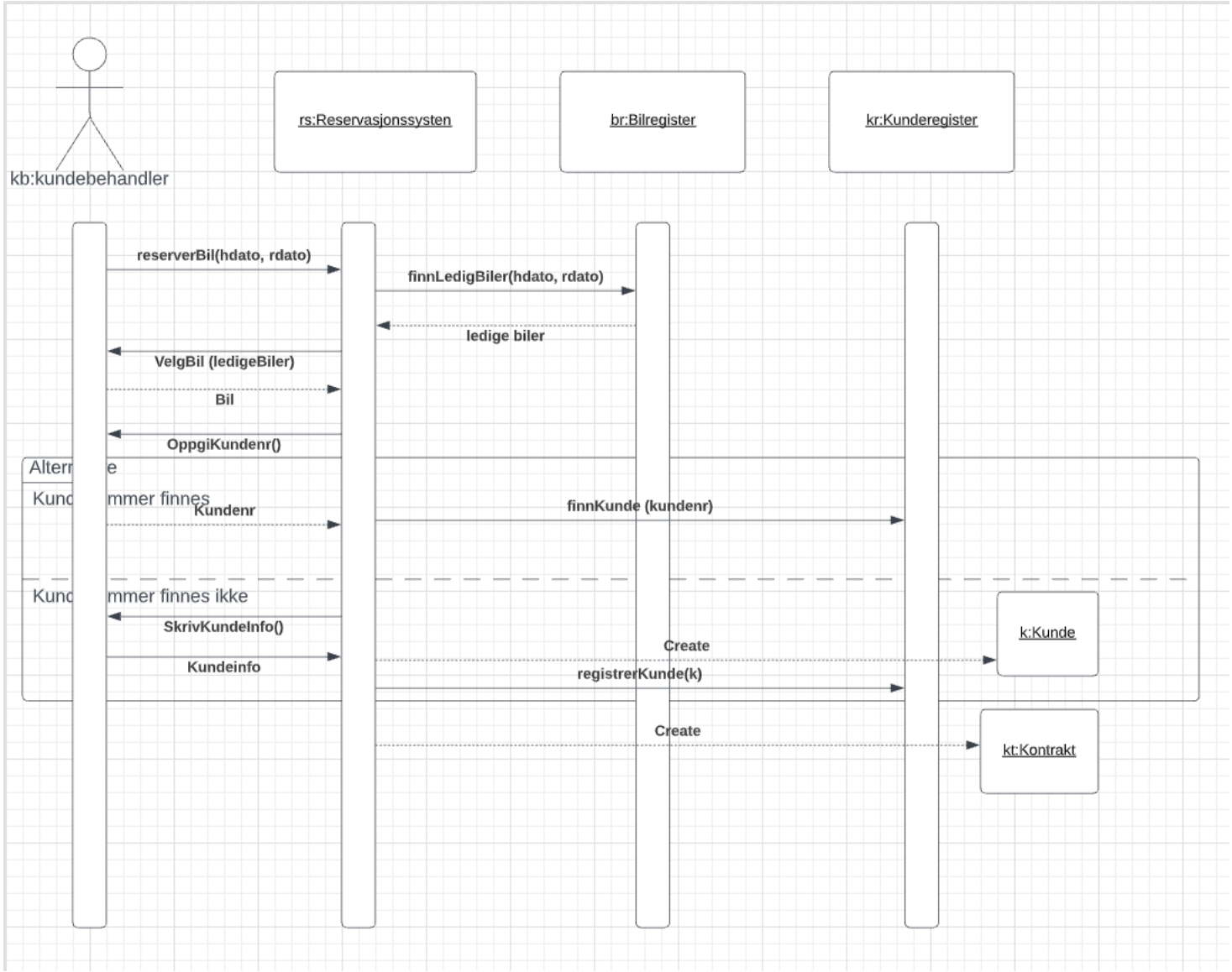


Kommentar: Kunne sendt med info til create Kvittering og Lodd (beløp og loddnr, + annen info)

Klassediagram – pante flasker



Sekvensdiagram - Reserver bil



Pseudo-kode – Reserver bil

```
class Reservasjonssystem {

        // Disse objektene kjenner vi fra før
        Bilregister br; Kunderegister kr; Kundebehandler kb;
        ArrayList <Bil> ledigeBiler;
        Bil bil; String kundenr; Kunde k;
        Kontrakt kt;

        // kundebehandler velger tidsintervall (hentedato og returdato)
        reserverBil (hdato, rdato) {

                // Systemet returnerer en liste over tilgjengelige biler innenfor gitte datoer
                ledigeBiler=br.finnLedigBiler(hdato, rdato);

                //Kundebehandler velger én av bilene
                bil=kb.velgBil(ledigeBiler);

                //Systemet ber om kundenr....
                kundenr=kb.oppgikundenummer();
                if (kundenr) {
                        //.... Finner kunden i systemet
                        k=kr.finnKunde(kundenr);
                } else  {
                        kundeinfo=kr.skrivKundeInfo();
                        k=new Kunde();
                        kr.registrerKunde(k);
                }
                //Lager ny kontrakt
                kt=new Kontrakt();
        }
}
```