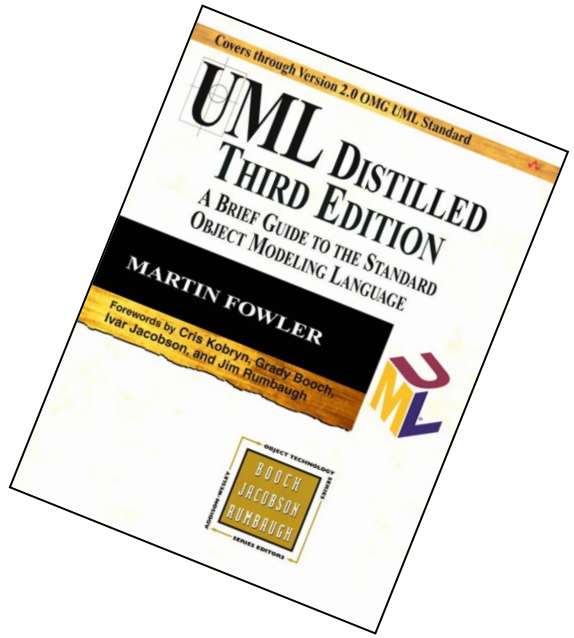
UML
Introduksjon, use-case-analyse og klassediagram

TTK4235



Kilder:

UML Distilled, 3rd ed., Martin Fowler

Læringsmål for UML

Kunnskap

- Grunnleggende kunnskap om *sentrale deler* av språket UML og hvordan det kan benyttes til å beskrive et tilpasset datasystem

Ferdigheter

- Kunne analysere, konstruere og implementere UML for enkle systemer
- Kunne benytte UML under systemplanlegging og –dokumentasjon

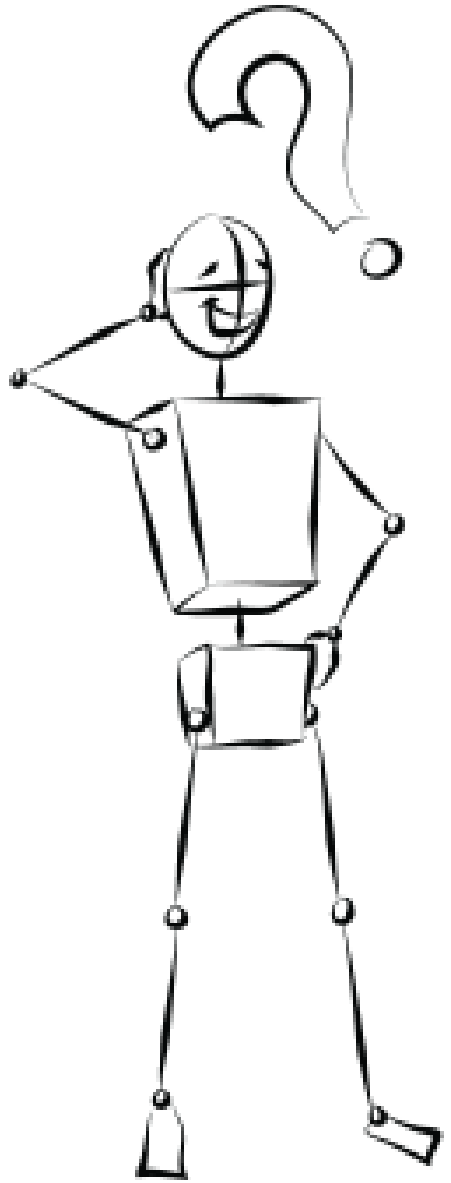
Generell kompetanse

- Kommunisere om datastyringssystemer og deres funksjonalitet både med spesialister og med systembrukere

Innhold

- Motivasjon
- Introduksjon til UML
- Use Case-begrepet (modellering av funksjonskrav)
- Klassediagrammet (moduloversikt)
- Eksempel fra egen forskning + rundetelleren

Motivasjon
– Hvorfor skal vi lære UML?



Det er behov for systematikk!

Systemutvikling koster (som alt annet)

Strukturert arbeidsmetode:

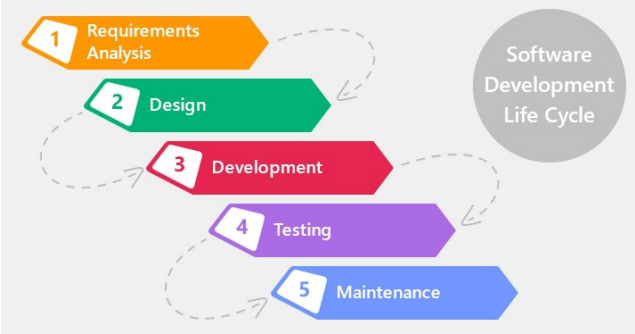
- Behandle forskjellige faser hver for seg

Visualisering av systemet f.eks. via UML kan være nyttig!

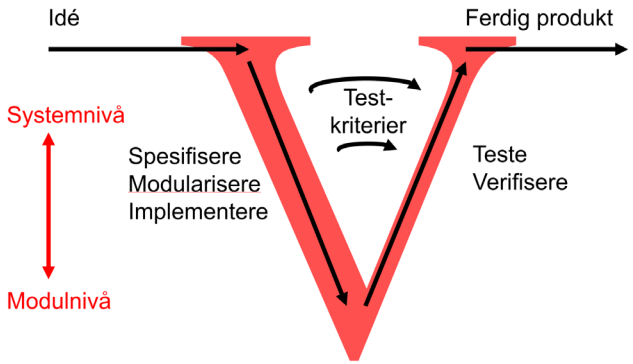
- Lettere mentalt og organisasjonsmessig

Gjør det lettere å oppnå:

- Robust og pålitelig programvare
- God kodekvalitet!
- God kommunikasjon/ dokumentasjon



www.xbsoftware.com



Smallsat, NTNU

Intro til UML
– the Unified Modeling Language

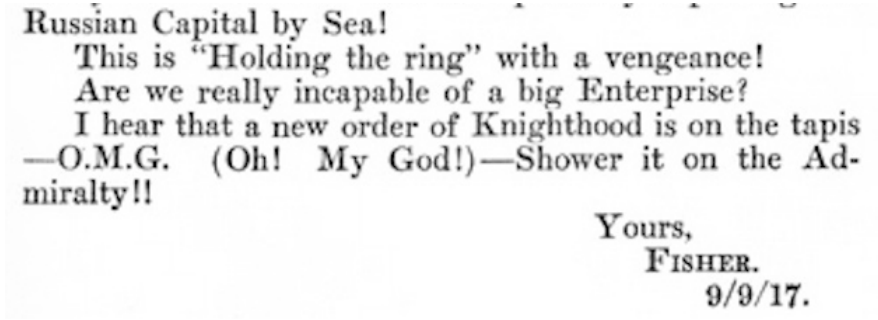
OMG: It's UML!!

UML = et "språk" for å beskrive og analysere systemer

- Representer en industristandard

Forvaltes av OMG (*The Object Management Group)*

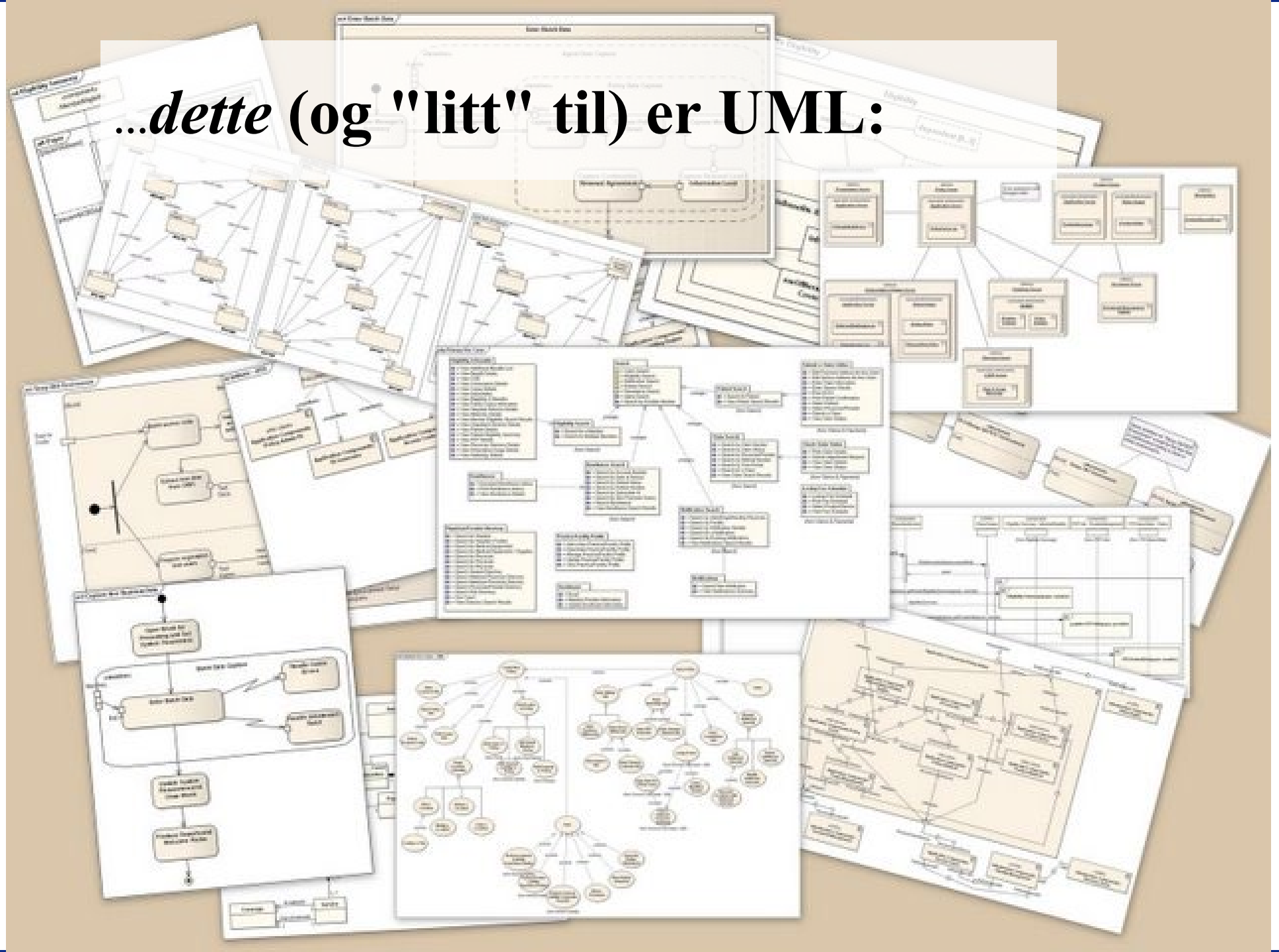
- Tilbyr sertifisering av ingeniører



Russian Capital by Sea!
This is "Holding the ring" with a vengeance!
Are we really incapable of a big Enterprise?
I hear that a new order of Knighthood is on the tapis —O.M.G. (Oh! My God!)—Shower it on the Admiralty!!

Yours,
FISHER.
9/9/17.

...*dette* (og "litt" til) er UML:



«UML-treet»

Forskjellige skjema som beskriver systemets:

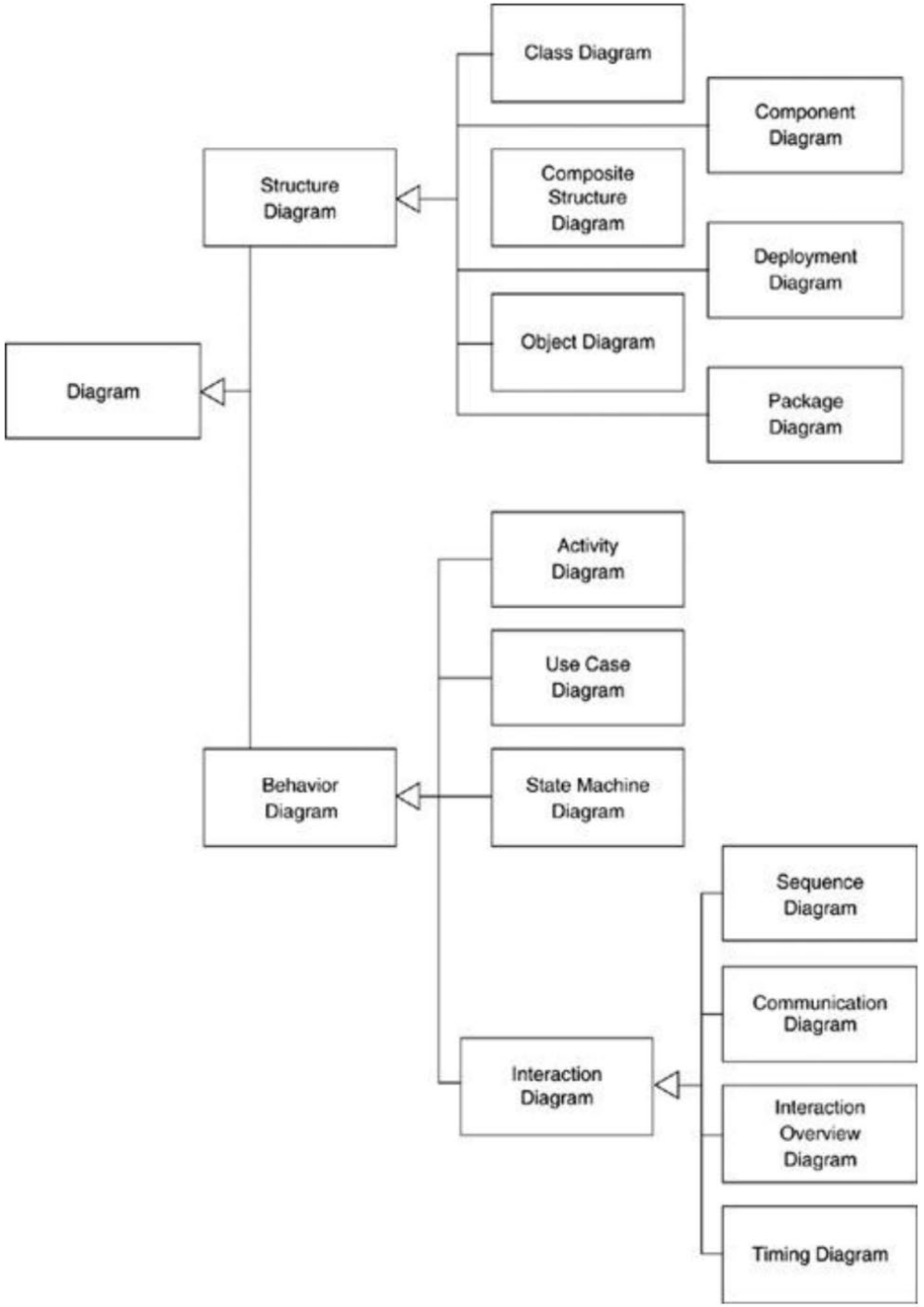
- Struktur
- Virkemåte

Kan brukes på forskjellig vis:

- Som programmeringsspråk
- Som «blueprint»
- Som skisse

Kan hende du «bruker» innholdet i noen av disse skjemaene ubevisst allerede

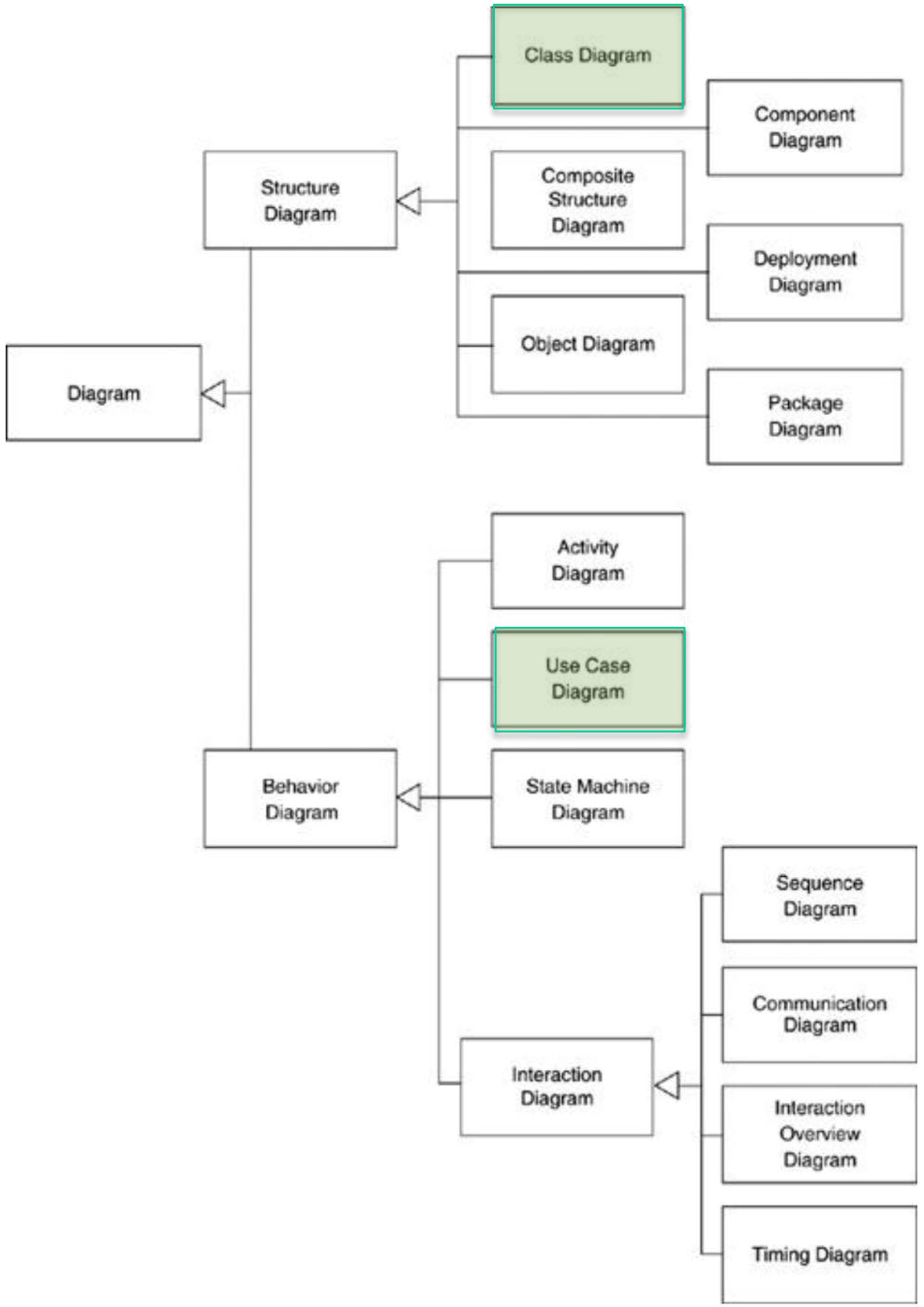
- Mentimeter



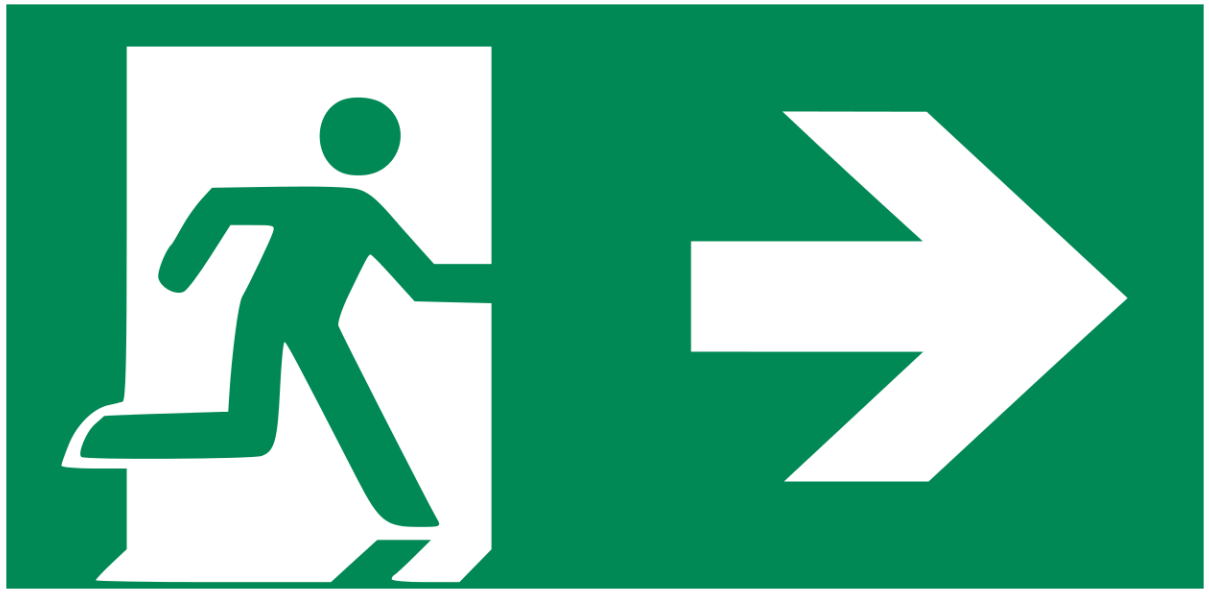
UML omfatter *mange* modelltyper, vi skal bruke 5:

Vi bruker…	**…for å**
Use case-modell	systematisere funksjonskrav
Klassediagrammet	beskrive modulenes sammensetning
Aktivitetsdiagram	beskrive arbeidsflyt data/kontroll
Tilstandsdiagram	beskrive (noen) modulers oppførsel
Sekvensdiagram	beskrive viktige hendelsessekvenser

Dagens tema


Class Diagram
Component Diagram
Structure Diagram
Composite Structure Diagram
Deployment Diagram
Object Diagram
Diagram
Package Diagram
Activity Diagram
Use Case Diagram
Behavior Diagram
State Machine Diagram
Sequence Diagram
Communication Diagram
Interaction Diagram
Interaction Overview Diagram
Timing Diagram

Use case-begrepet

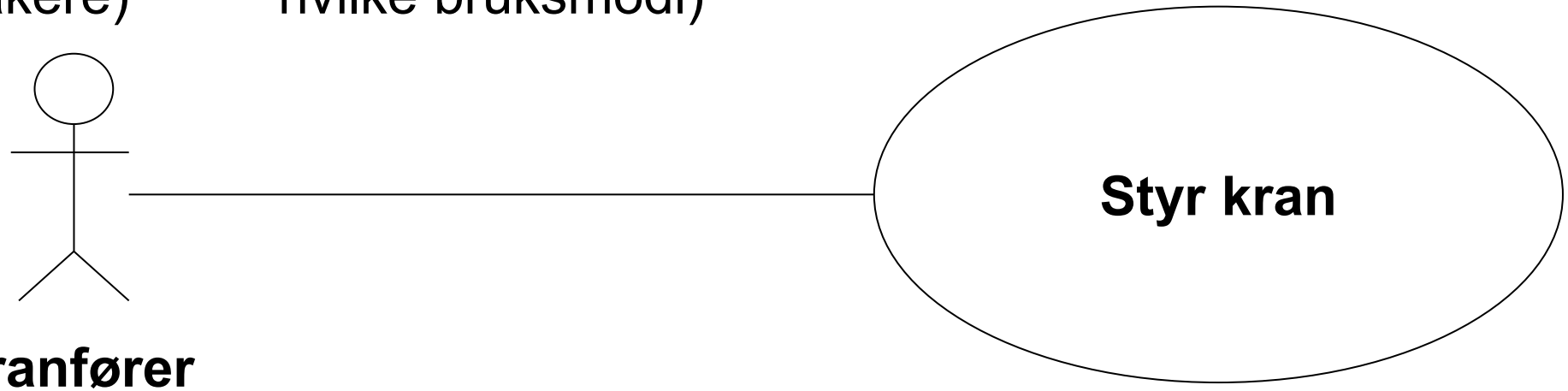


Use case-diagrammet

Actor
(rollene
til ytre
deltakere)

Association
(hvilke roller
som inngår i
hvilke bruksmodi)

Use case
("bruksmodus", måte
å forholde seg til systemet på)

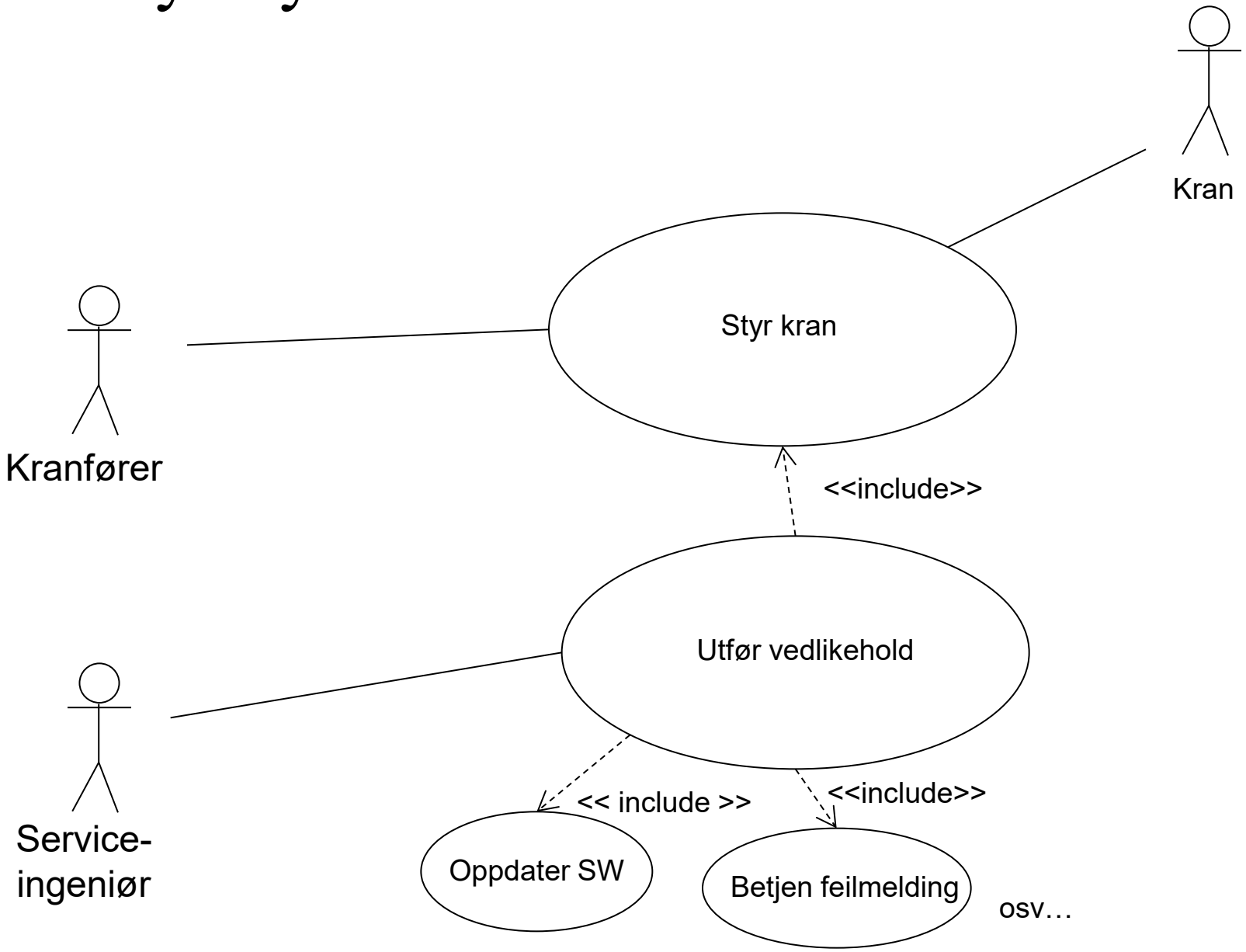


Use case = ”bruksmodus”

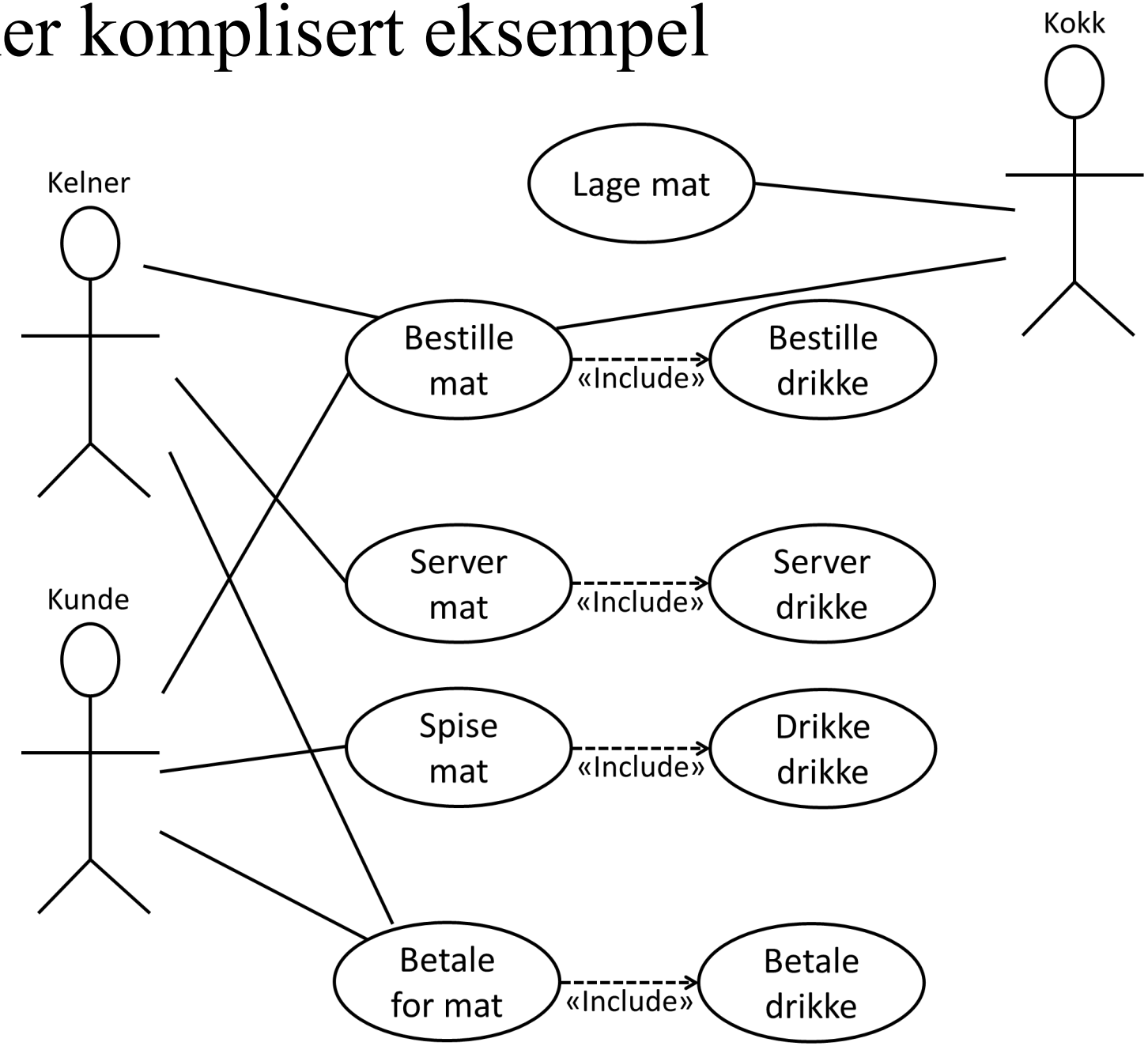
- Et tilpasset datasystem tilbyr et antall funksjoner som er aktuelle for ulike **formål**, til ulike **tider** og for ulike **aktører**
- Hvert **use case** er én slik ”bruksmodus”

Summen av alle use cases
=
hele systemets funksjonalitet

Eksempel: Styresystem for kran



Et litt mer komplisert eksempel



Hvordan **ikke** bruke use case-diagrammet

- Ikke bruk masse tid på å lage et ”avansert” diagram!

KISS:
Keep **I**t **S**imple, **S**tupid!

- Alt annet er bortkastet tid, for use case-diagrammet er bare en "innholdsfortegnelse"!

Use case-beskrivelsen
-- viktigere enn diagrammet!

Pre-conditions:

Vilkår som må være tilfredsstilt for at use case't kan starte

Trigger:

Hendelsen som starter use case't

Scenarier:

Hvordan actors og system interagerer når systemet er i drift

- **Hovedscenario (Min success scenario):** Når alt er "normalt"
- **Utvidelser (Extensions):** Alle avvik fra hovedscenariet.

Guarantee:

Det systemet vil garantere er oppfylt ved avslutningen av use case't

- **Success guarantee:** Det som garantert er oppfylt etter et *vellykket* scenario
- **Minimal guarantee:** Det som er oppfylt etter et hvilket som helst scenario.

Eksempel på use case-beskrivelse: ”Styr kran”

Styr kran

Precondition: Kranfører i setet, kranen tikoplet strøm
kranen i ro

Trigger: dødmannsknapp aktivert

Suksess-scenario:

1. Kranfører beveger joystick
2. Kranens motorer kjøres med hastighet gitt av joystick-utslag
3. Dødmannsknapp slippes
4. Kranen stopper.

Utvidelser:

1a: Dødmannsknapp slippes
1a.1 Hopp til punkt 4

2a: Dødmannsknapp slippes
2a.1 Hopp til punkt 4

Suksessgaranti:
Som Precondition

Minimal garanti:
Kran i ro

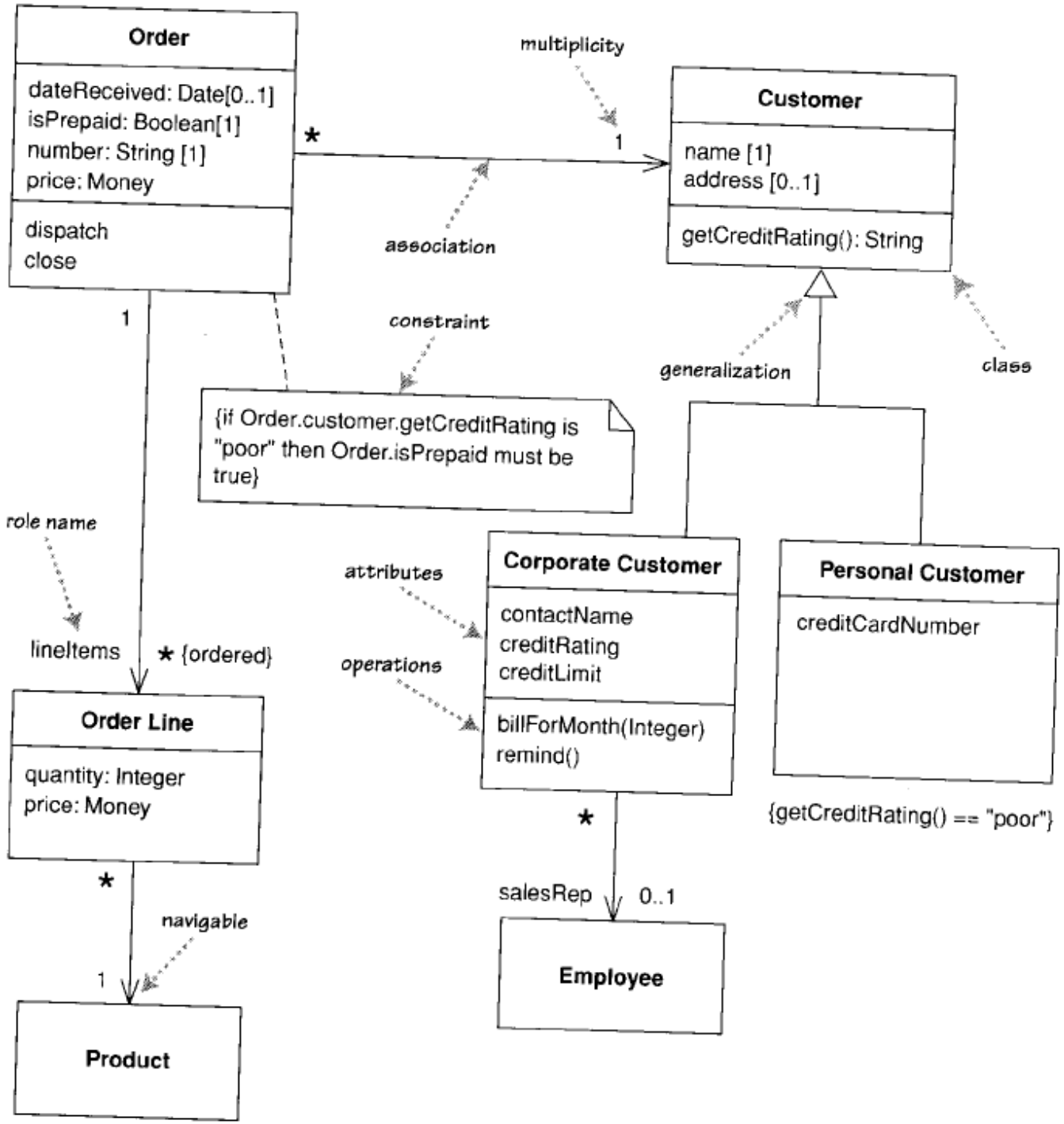
Hvem har nytte av denne informasjonen?

Klassediagrammet

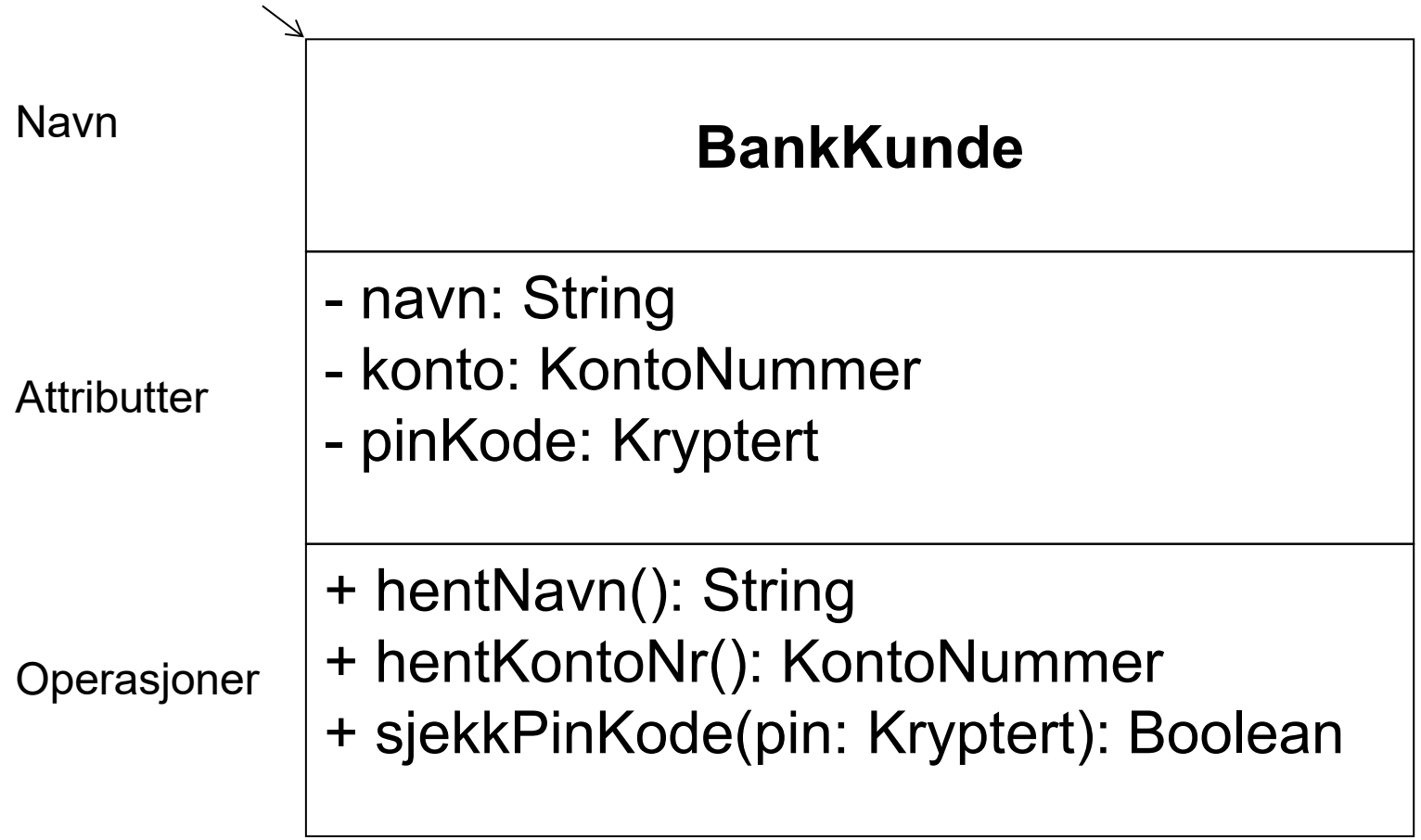
Moduloversikt

Gruppering av systemet i delsystemer/moduler

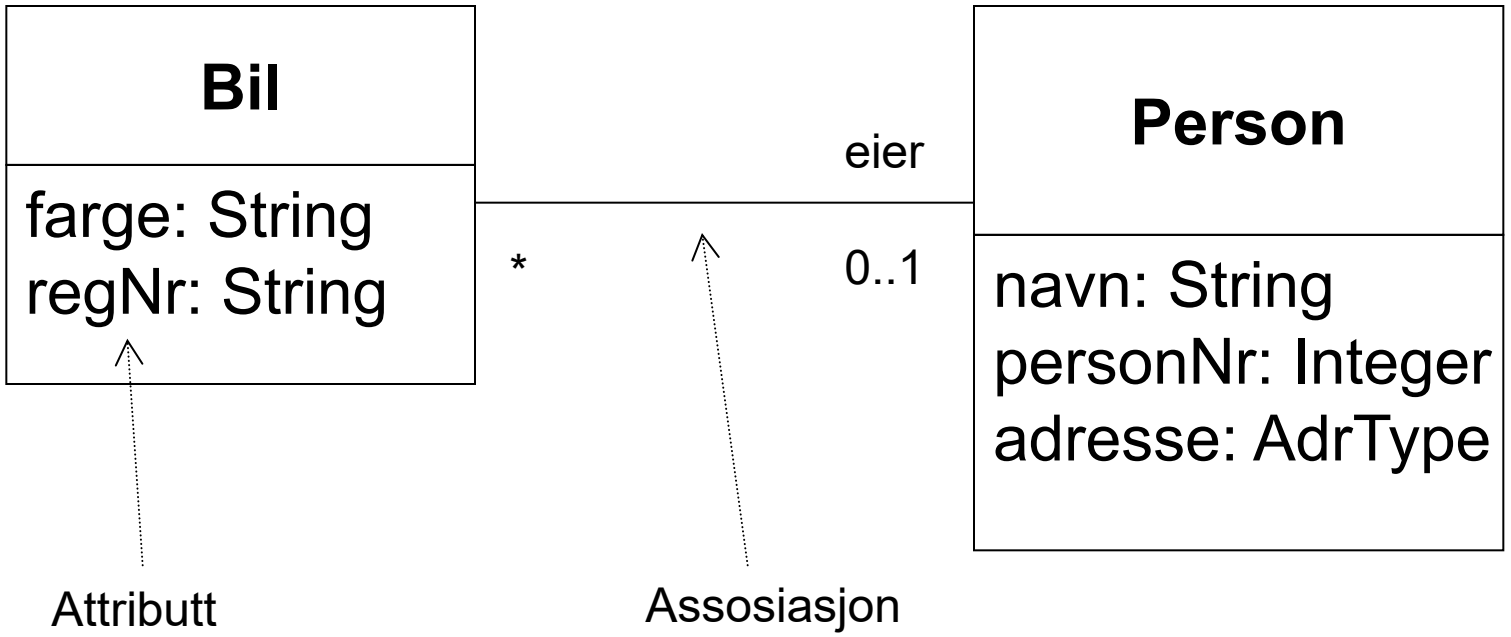
Klassediagrammet



Klasse (bare øverste seksjon er obligatorisk)



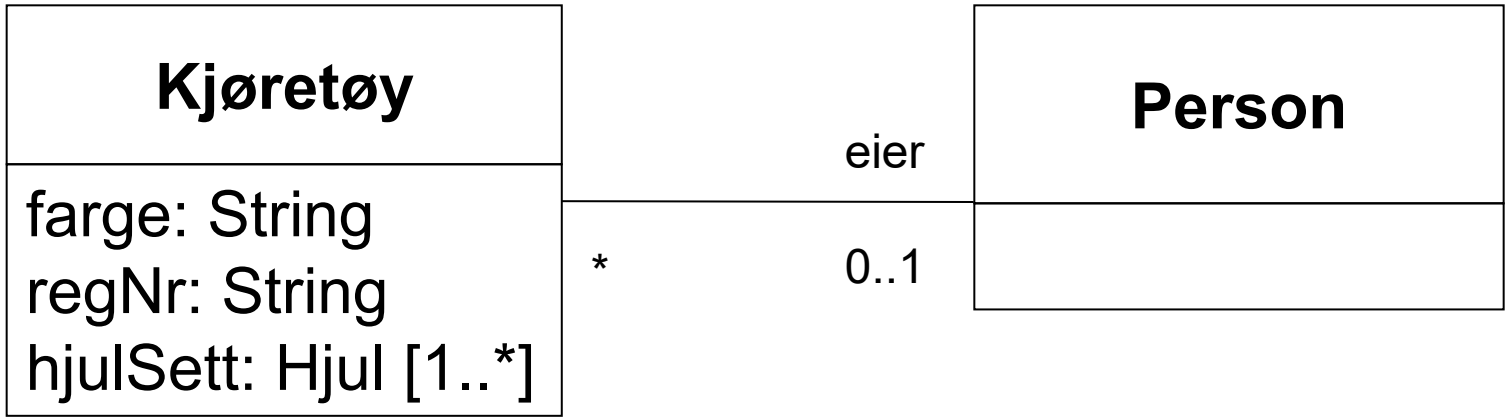
Klassens egenskaper: Attributt vs. assosiasjon



Notasjon - Multiplisitet

1	Nøyaktig én
0..2	0, 1 eller 2
*****	0 eller flere

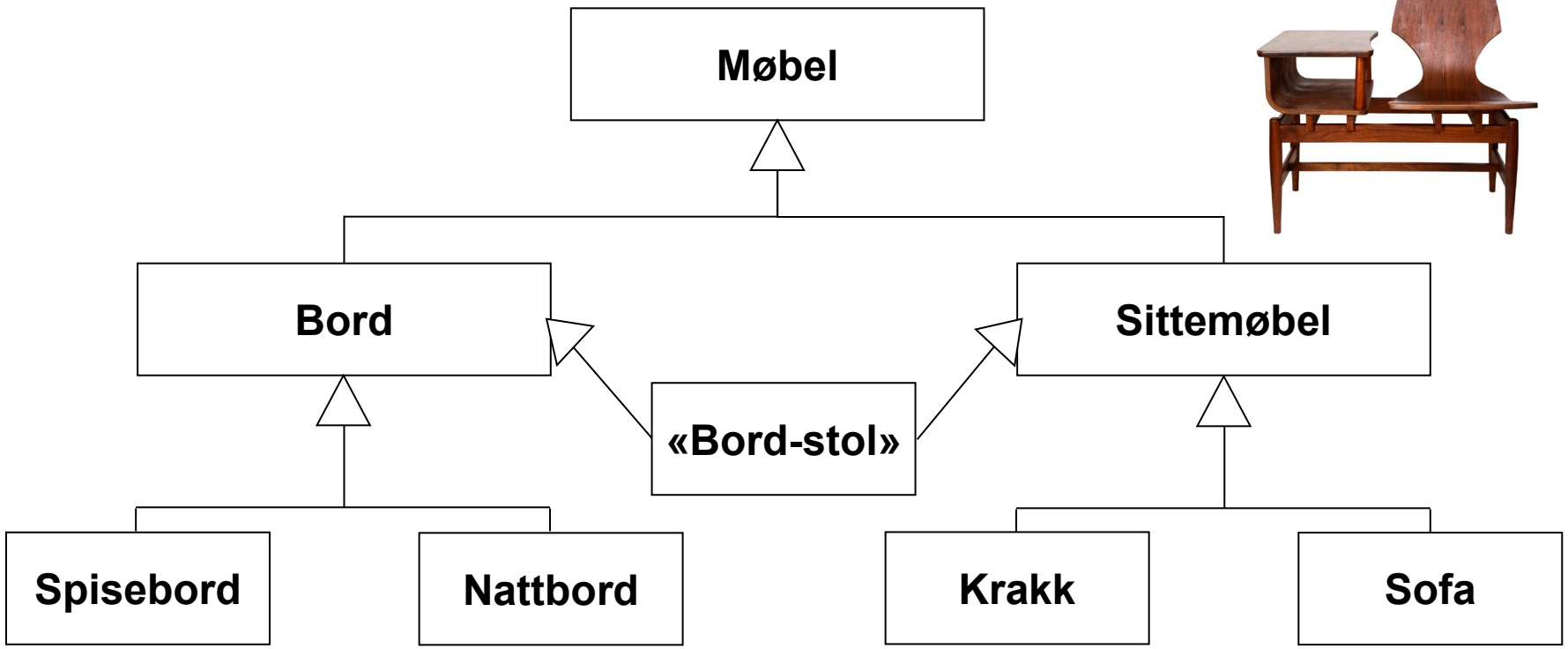
Ingen multiplisitet angitt => **1**



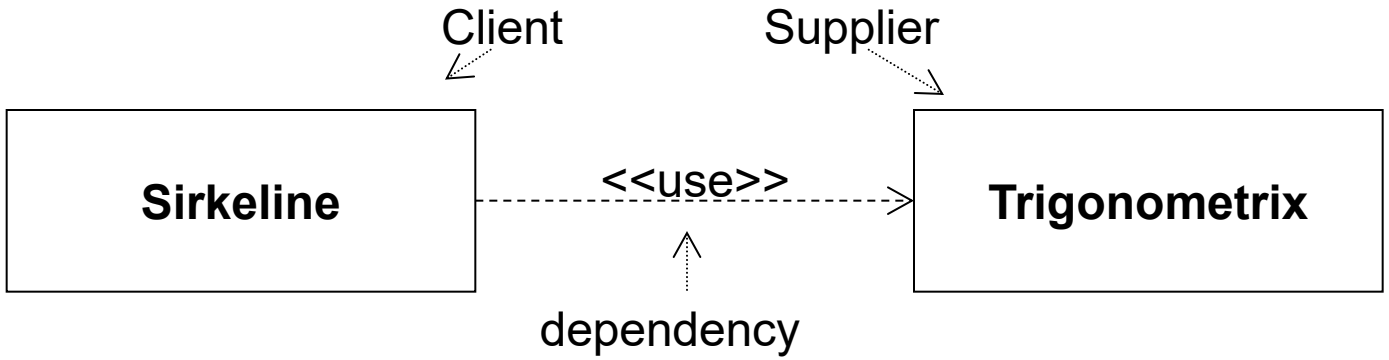
Notasjon - Generalisering

"Arv" (eks):

Alle Sittemøbler har Møbel sine egenskaper, men de kan ha egne egenskaper i tillegg.



Avhengighet (Dependency)



Assosiasjoner, generaliseringer osv. medfører implisitt *avhengighet*

Men avhengighet kan også være eksplisitt definert

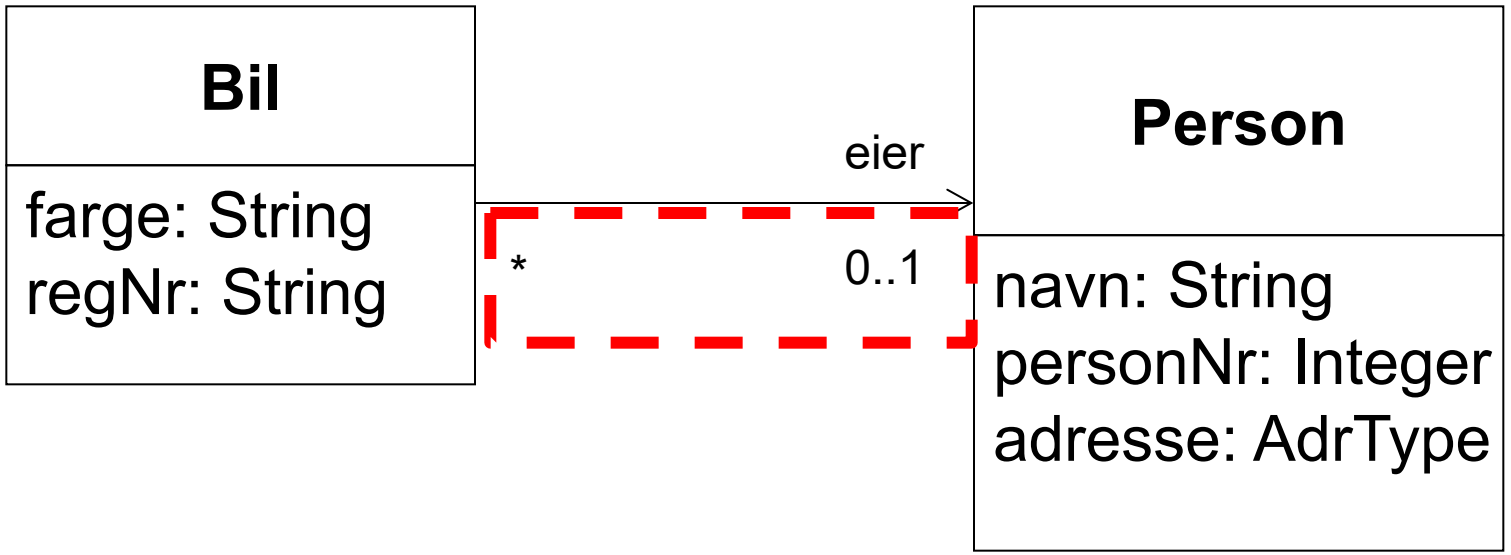
Scenario: Sirkeline lager sirkler. Til dette trenger hun funksjonalitet som Trigonometrix tilbyr.

- Hvis Trigonometrix forandrer oppførsel, kan dette få følger for Sirkeline slik at hun også må endre seg.
- Trigonometrix er derimot helt uavhengig av Sirkeline.

Begrensninger/constraints

Constraint Rules: I UML kan du skrive hva som helst i diagrammet, på et hvilket som helst språk, bare du setter det i krøllparentes:

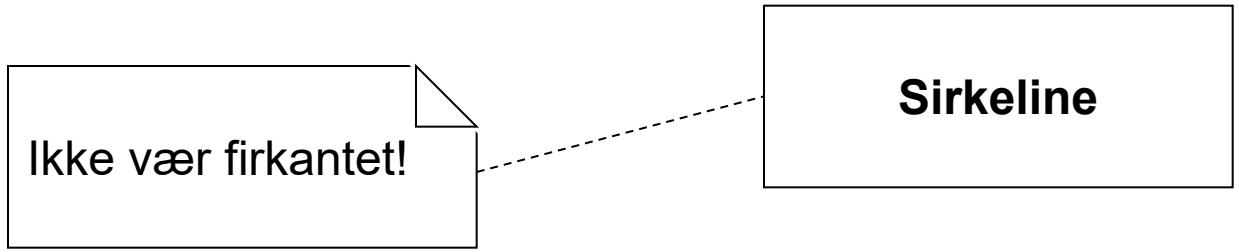
{korrekthetssjekk: alder>0 år}



Hva så med alt det andre...?

Finnes mye ekstra som kan defineres innenfor UML, men som ofte er for spesielt interesserte

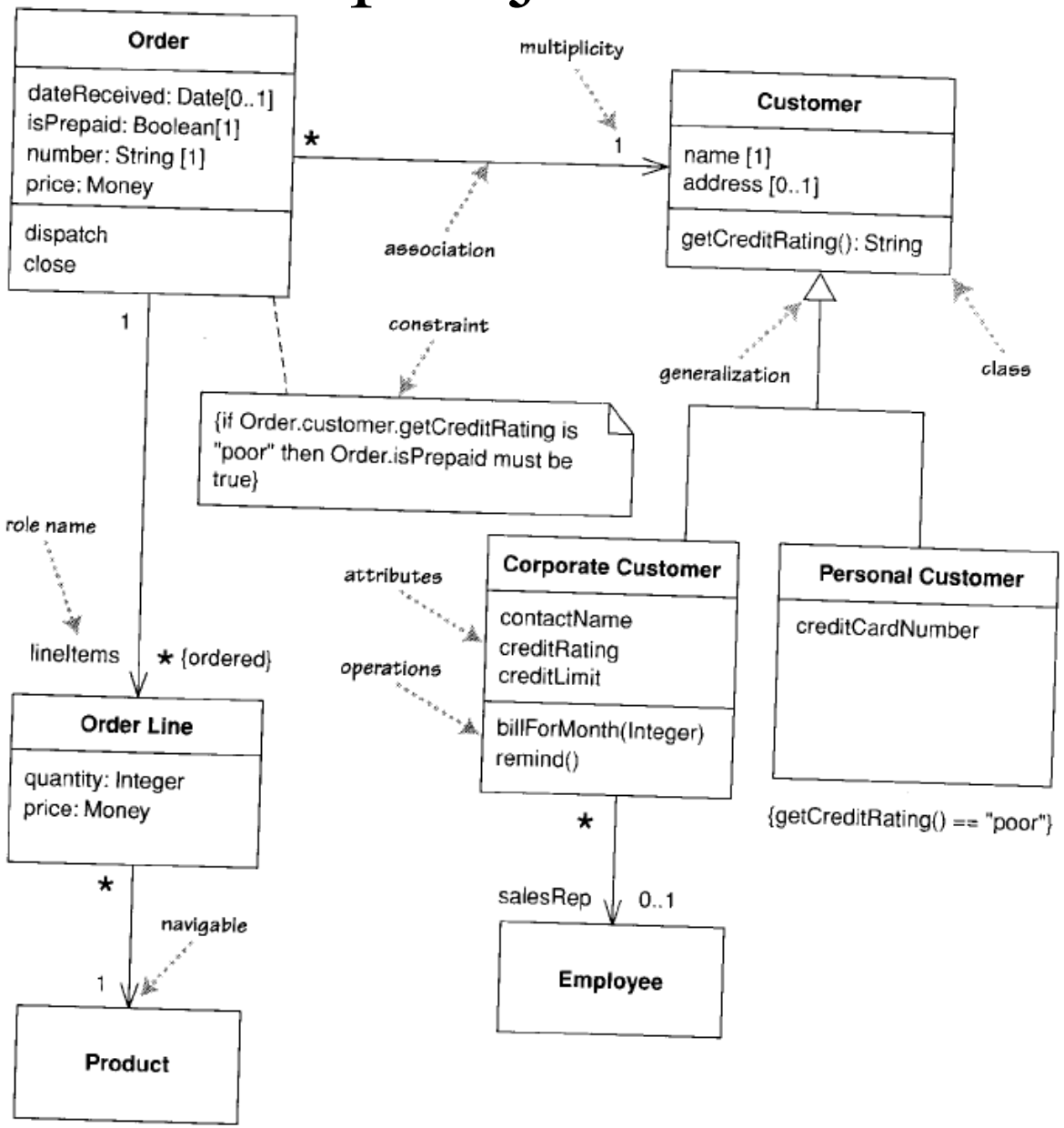
Dessuten har vi ”**kommentarboksen**”:



Uansett: ikke gjør ting for komplisert!

Bruk bare den notasjonen som er nødvendig

Klassediagrammet: repetisjon



Case: individbasert modell av fisk

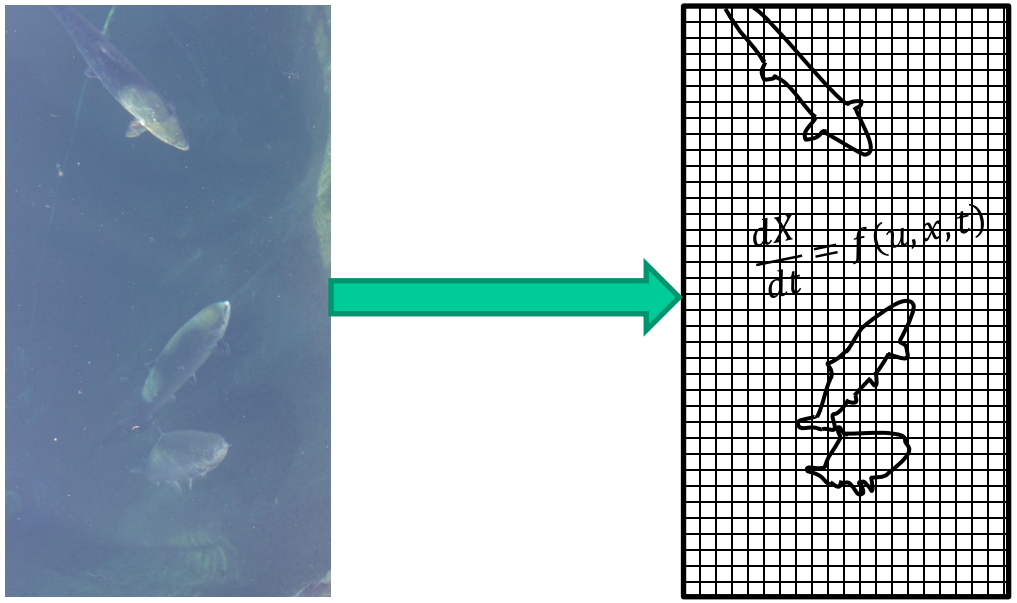
Mål: vi ønsker å simulere atferd og vekst for oppdrettsfisk i merd

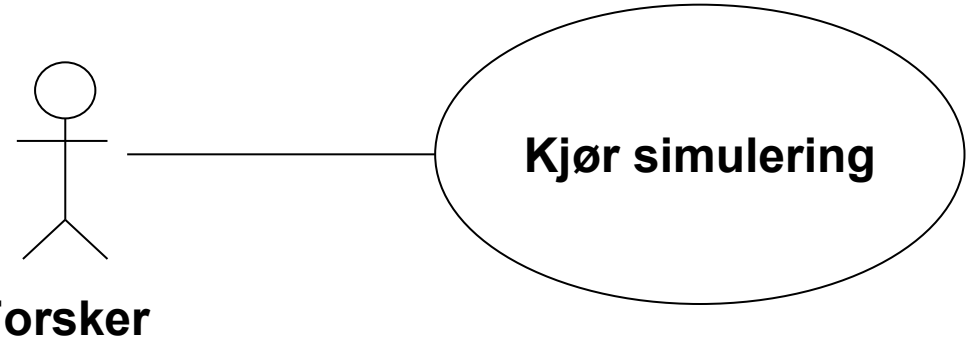
Krav:

1. **Modellere individfisk**
2. **Inputs: merd, strømning, temperatur, lys, fôr (NB: samme miljø for alle individ!)**
3. **Skal kunne håndtere forskjellige fiskearter**
4. **Både fisk og miljø varierer med tid**

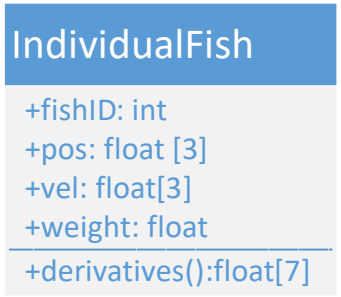
Use-case: ikke veldig komplisert!

Men diskuter klassediagram!

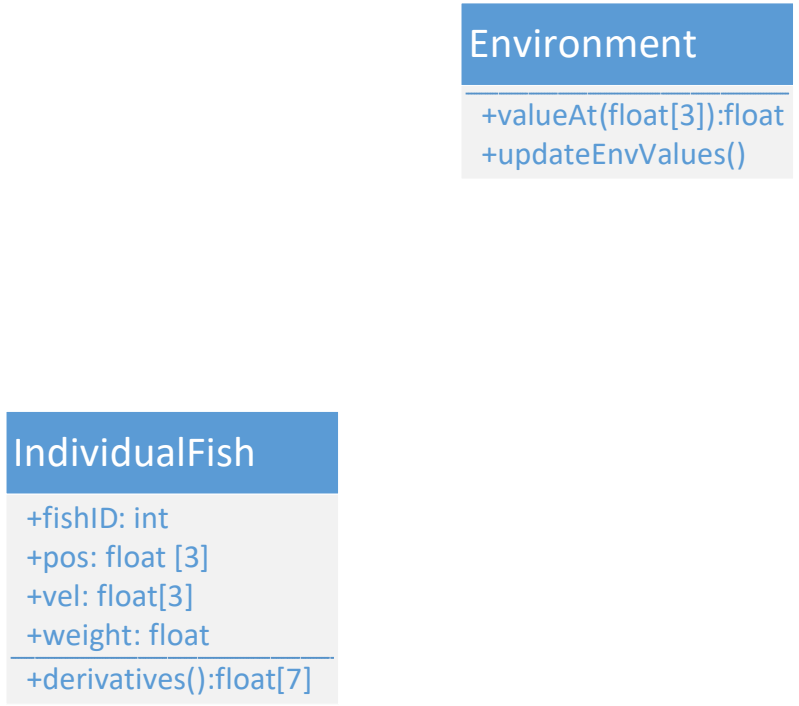




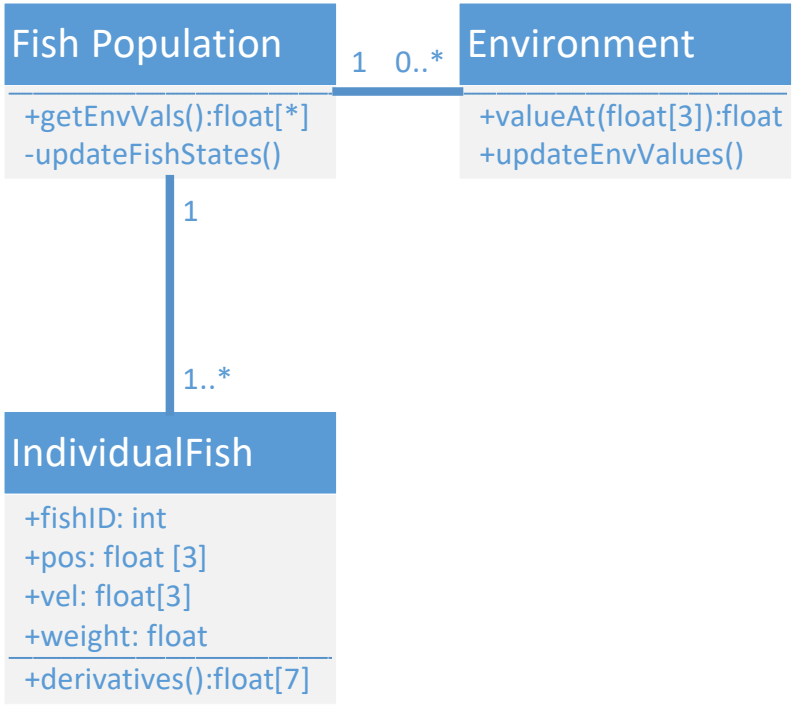
Klassediagram for fiskemodell



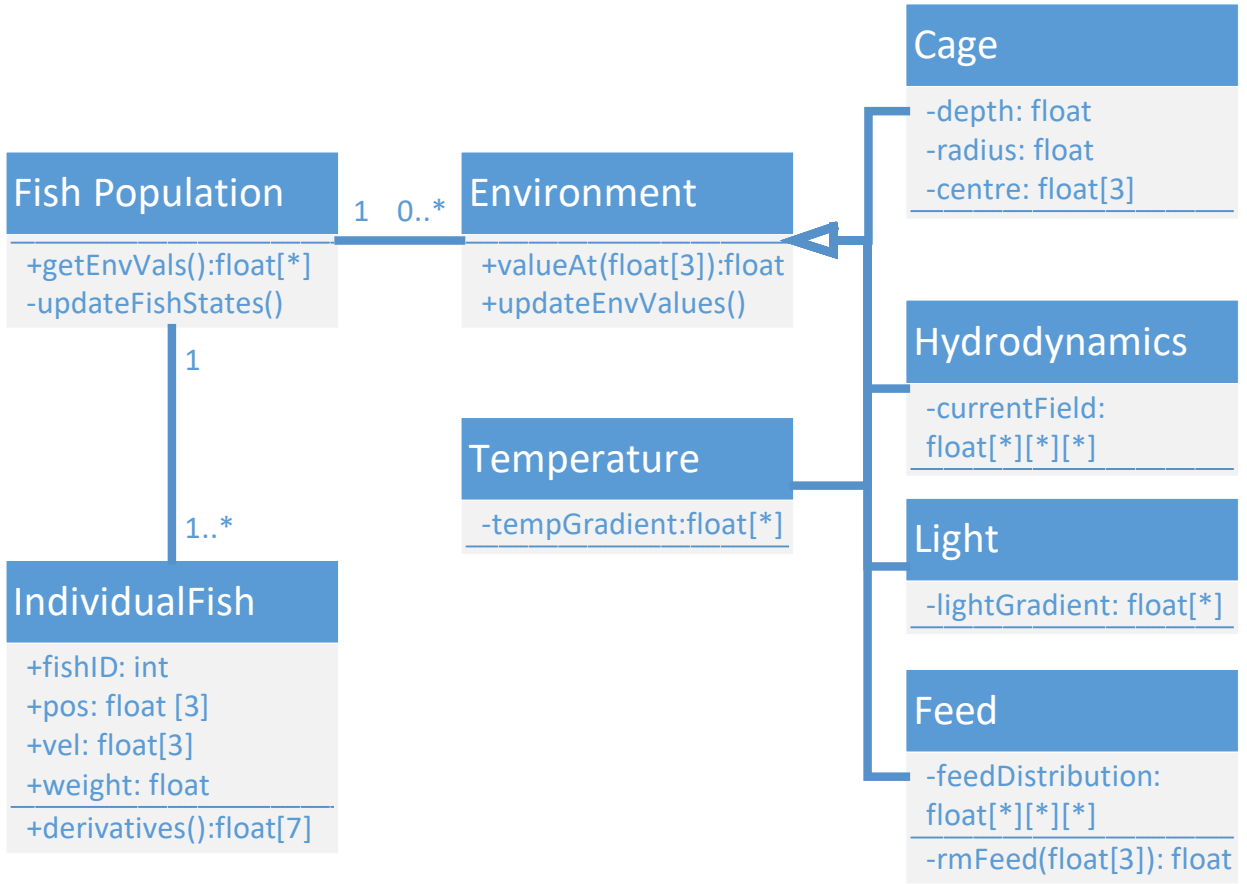
Klassediagram for fiskemodell



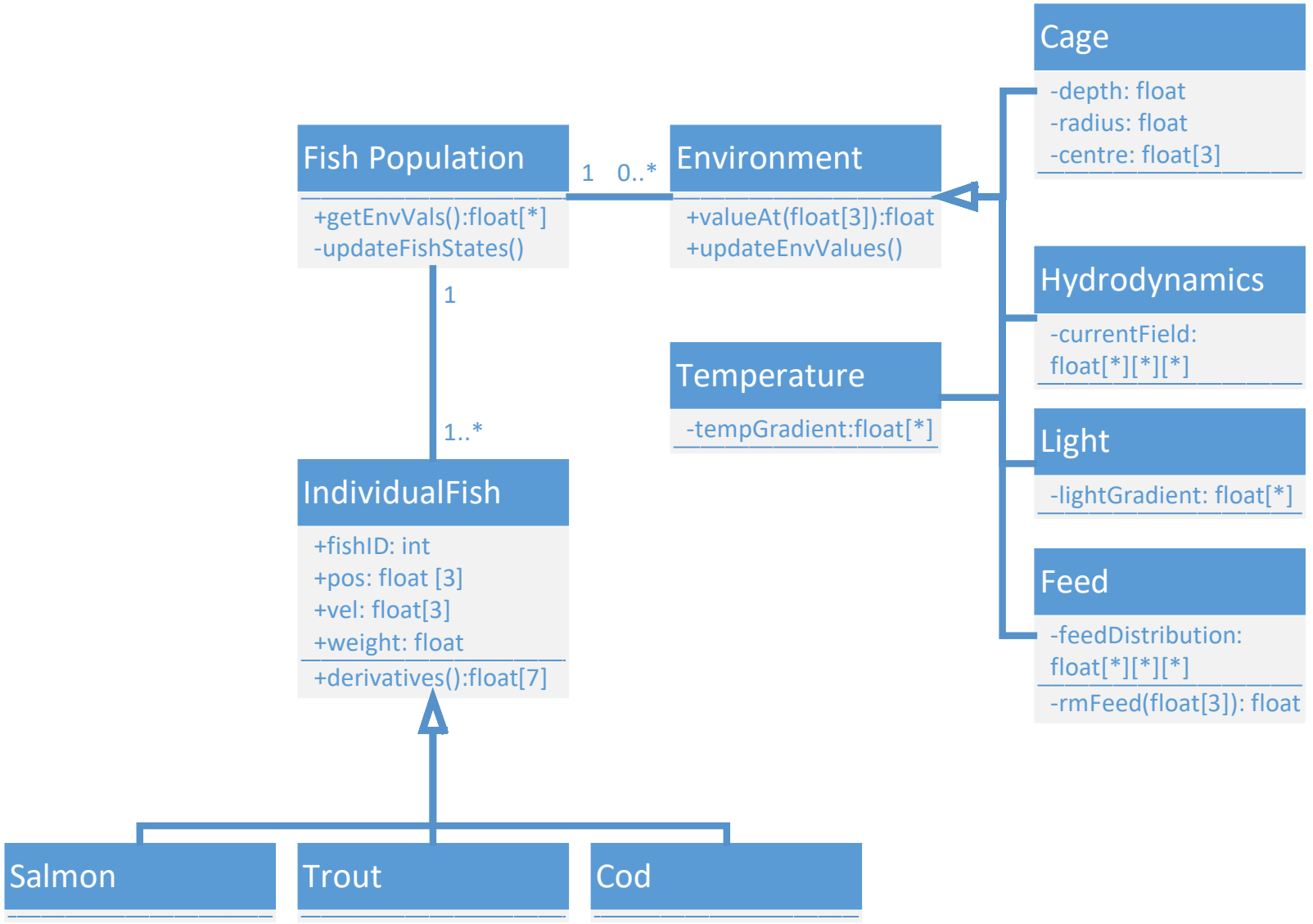
Klassediagram for fiskemodell



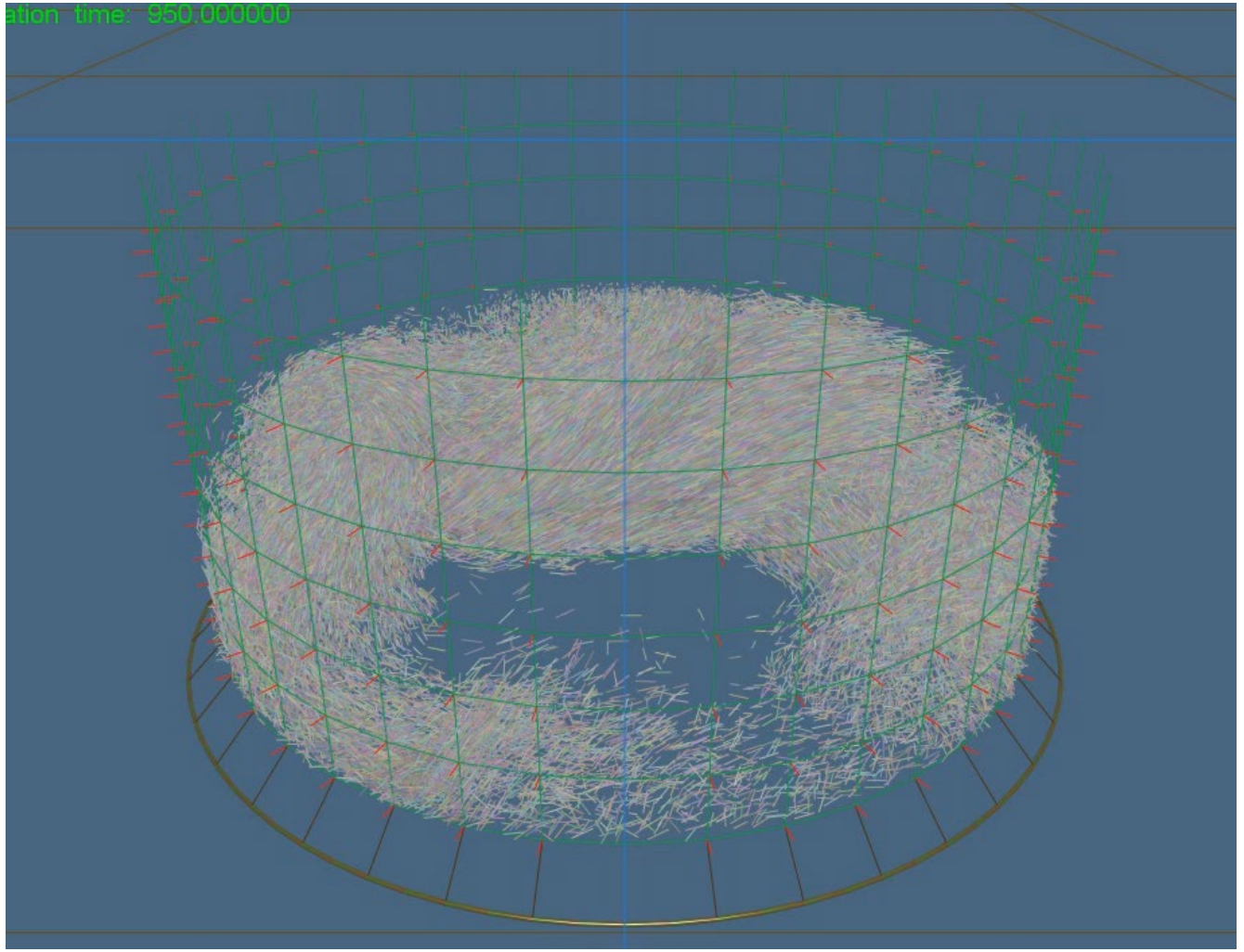
Klassediagram for fiskemodell



Klassediagram for fiskemodell



Resultat til slutt



Systemutvikling med UML
"Rundetellereksemplet"

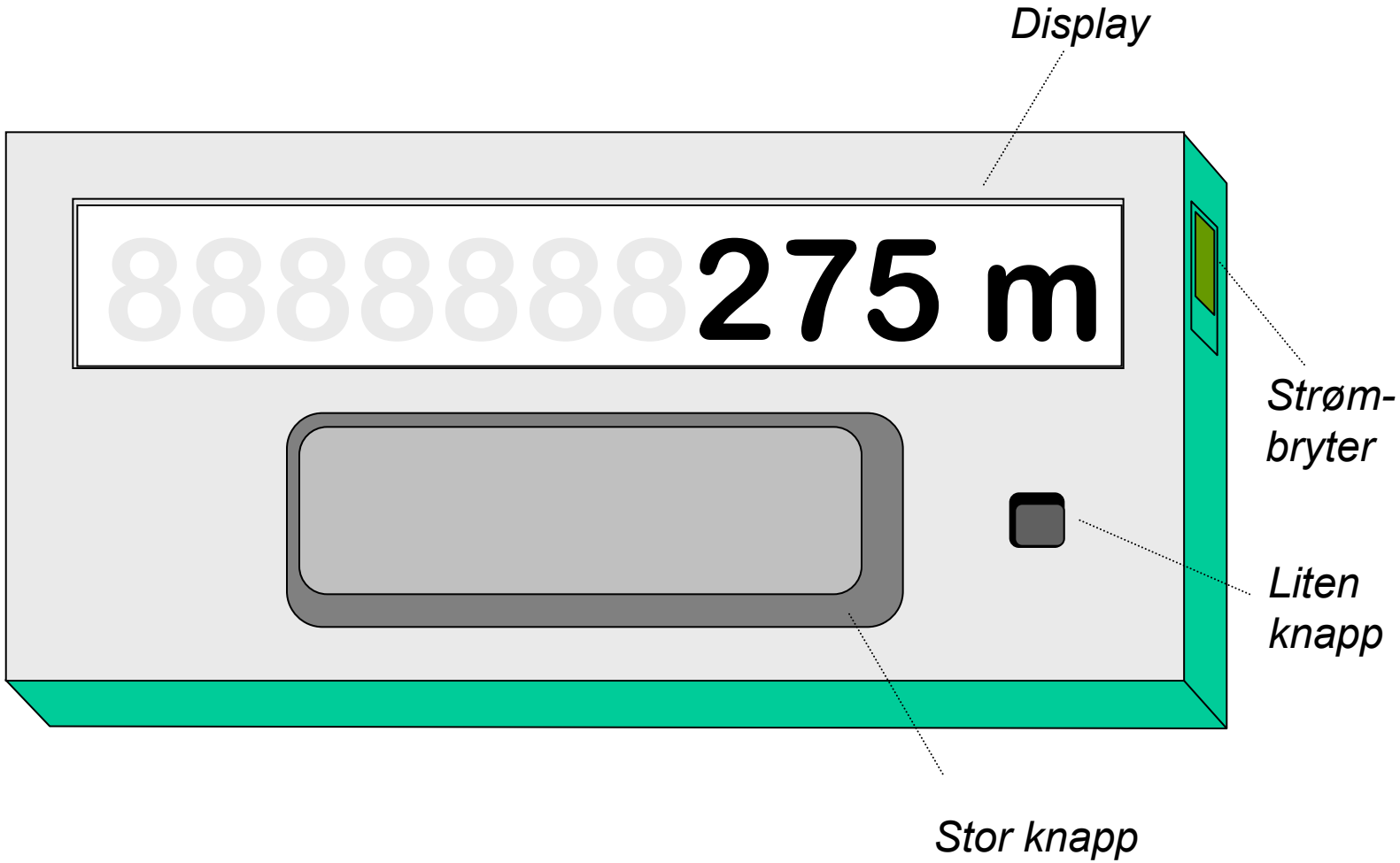
Rundetelleren

Eksempel på systemmodellering med UML:

- Use case-analyse
- Klassediagram
  (modulenes relasjoner og grensesnitt)

- Tilstandsmaskin (modulen(e)s oppførsel)
  - Kokebok for konstruksjon av tilstandsmaskin
- Supplerende diagrammer som sekvensdiagram

Eksempel: ”Tripteller” for svømmebasseng



Funksjonsspesifikasjon (prosaversjon)

Triptelleren slås på med strømbryteren og festes med sugekopper på bassengveggen der treningsøkten starter.

Telleverket nullstilles ved at den store knappen trykkes inn i 3 s.

Ved normal bruk vises svømt distanse kontinuerlig på displayet. Hver gang svømmeren vender ved den bassengenden der telleren er festet, trykkes det kort (< 3 s) på den store knappen. Når dette skjer økes svømt distanse med to ganger bassenglenden (dvs. lengden av én runde).

Dersom den lille knappen trykkes inn med en spiss gjenstand, kan bassenglenden stilles inn ved å trykke på den store knappen.

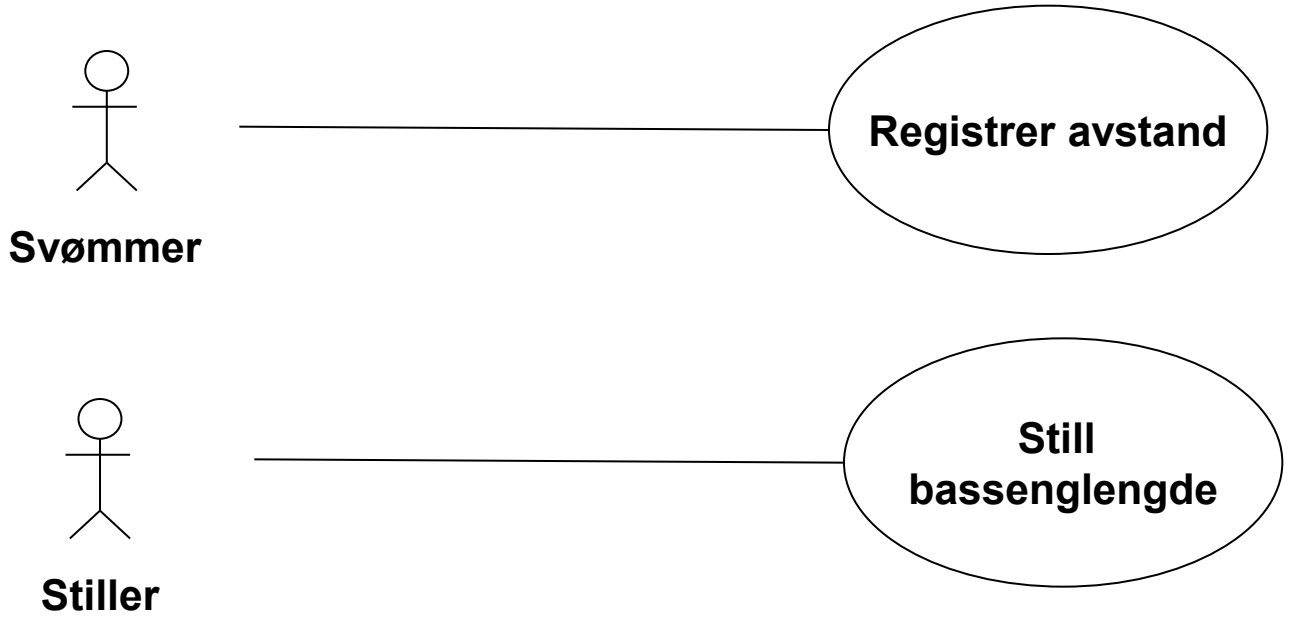
Displayet veksler da mellom tallene {12.5, 25, 50, 12.5, ...}, nytt tall for hvert trykk. Når det er gått 3 s siden siste tastetrykk, lagres det sist viste tallet som ny bassenglengde og systemet er klart til normal bruk igjen.

Når treningsøkten er over, slås telleren av med strømbryteren.

For enkelthets skyld antar vi at alle lagrede data (bassenglengde og svømt distanse) automatisk huskes selv om triptelleren slås av og på igjen.

Use case-diagram

Funksjonsspesifikasjonen antyder to use cases og to aktører:



Use case-beskrivelse:

Registrer avstand («normal bruk»)

Precondition: Riktig bassenglengde er stilt inn,
systemet er slått på.

Trigger: Den store knappen trykkes inn

Hovedscenario:

1. Den store knappen slippes igjen før det har gått 3 s
2. (2 x bassenglengden) legges til svømt distanse
3. Svømt distanse vises på displayet

Utvidelse:

1a: Den store knappen holdes inne i 3 s

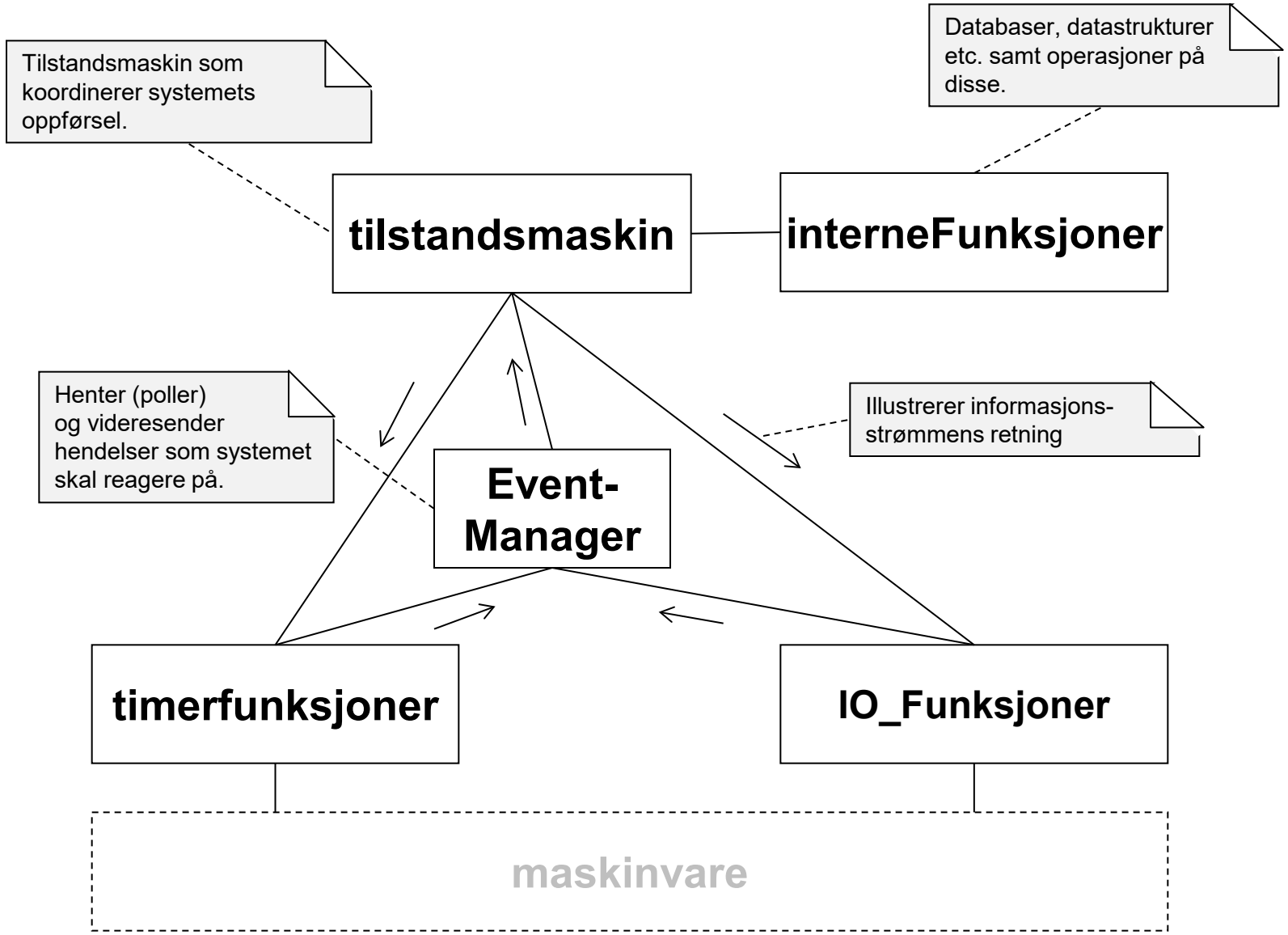
.1: Svømt distanse nullstilles

.2: Returnerer til hovedscenariets trinn 3.

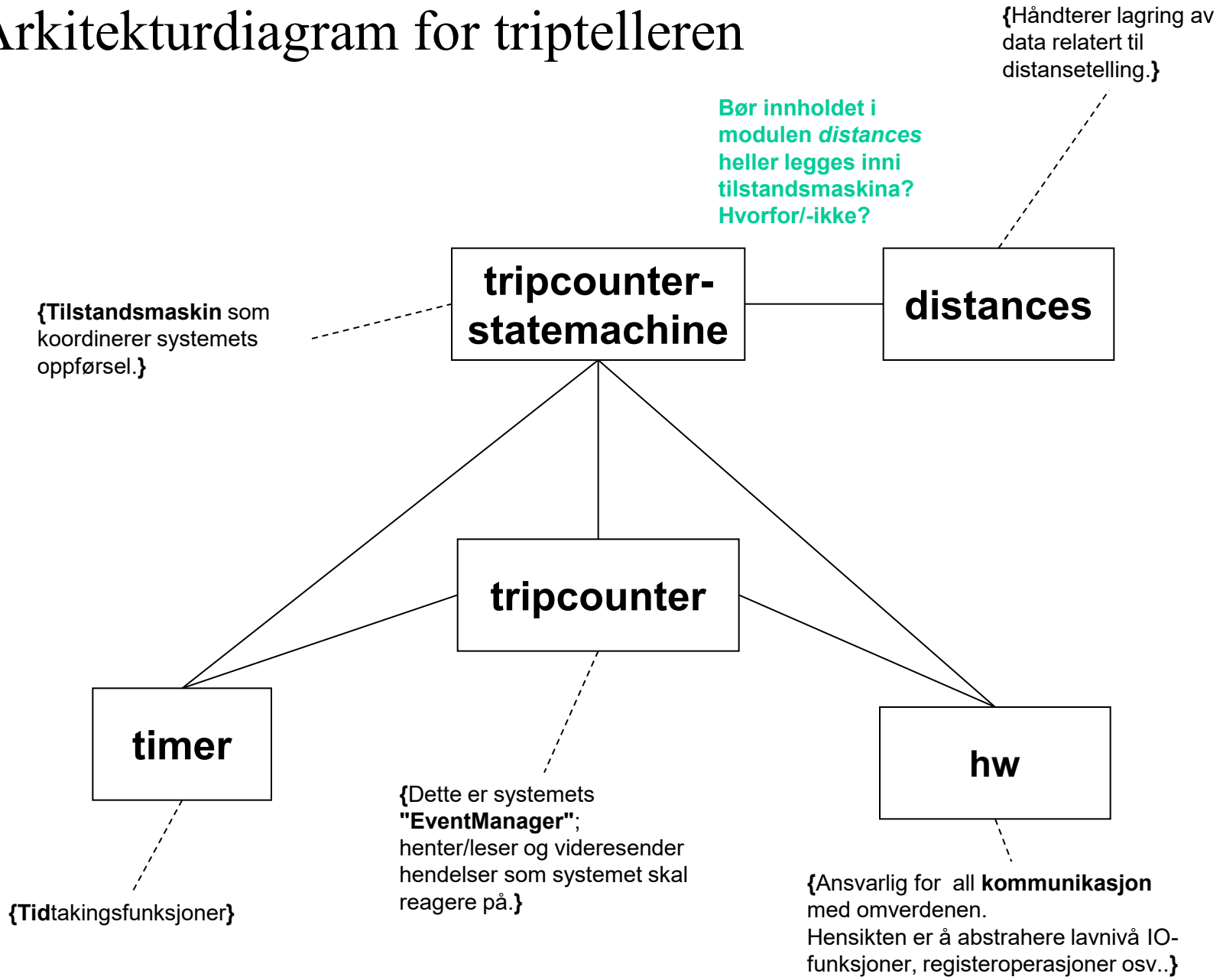
Garanti:

Svømt distanse vises på displayet. *(kanskje overflødig)*

Generisk arkitektur for hendelsesdrevne systemer (vi skal bruke denne)



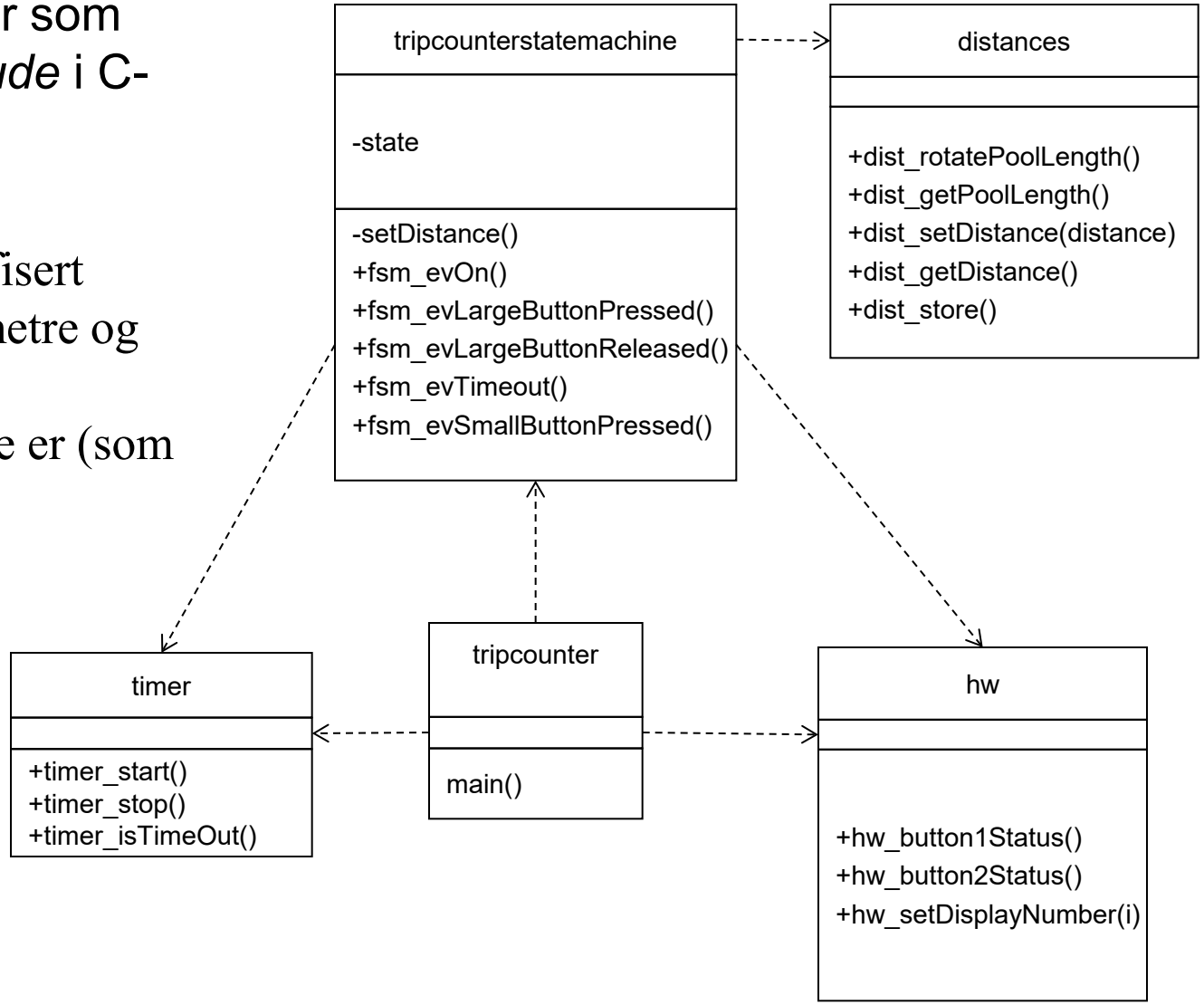
Arkitekturdiagram for triptelleren



Klassediagram for triptelleren

- Avhengighetene (stiplede piler) tilsvarer moduler som inkluderes med *#include* i C-kildekoden.

- Vi kunne også ha spesifisert datatype for både parametre og returverdier direkte i klassediagrammet. Dette er (som mye annet) valgfritt…



Neste gang

- Tilstandsdiagram
- Sekvensdiagram
- Aktivitetsdiagram
- Fortsettelse på rundetellereksempelet