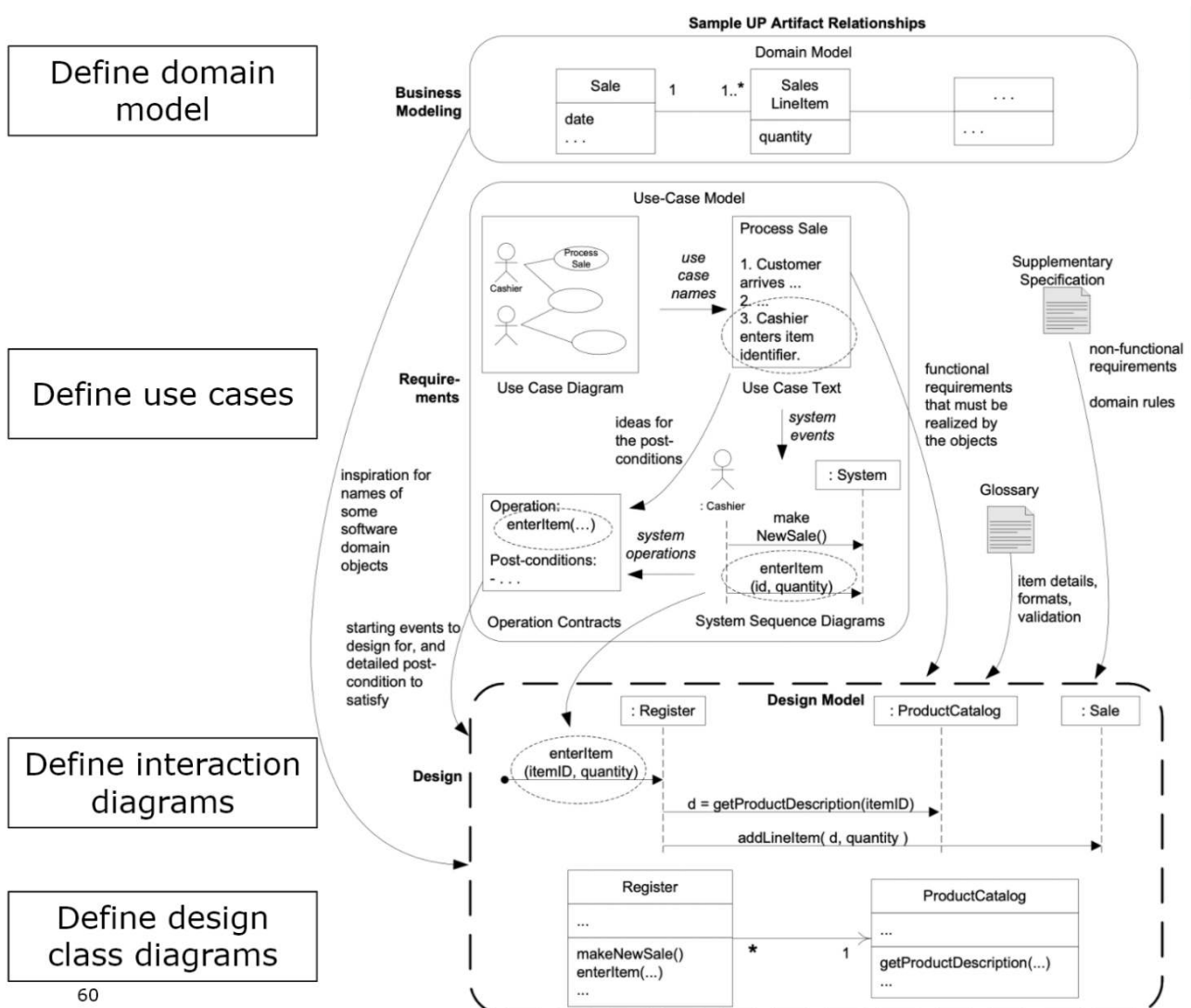




## Samlede noter

Udviklingsmetoder til IT-systemer (Danmarks Tekniske Universitet)

- Planlægge, styre og gennemføre et **mindre softwareprojekt** i projektgrupper
- Definere og beskrive almindelige **UML diagrammer**
- Anvende og forklare en moderne software **udviklingsproces**
- Bruge UML til at **modellere** softwaresystemer
- Udarbejde en **kravspecifikation**
- Udarbejde **design** for programmer
- **Udvikle** mindre programmer på baggrund af udarbejdet design
- Udvikle **programdokumentation**
- **Evaluere** kode og dokumentation vha. reviewteknikker
- Udarbejde rapport, der dokumenterer og vurderer projektets færdige produkt (**produktreport**)
- Udarbejde rapport, der dokumenterer, evaluerer og reflekterer over projektforløbet (**procesreport**)
- Kunne skrive tekniske rapporter.



# Indhold

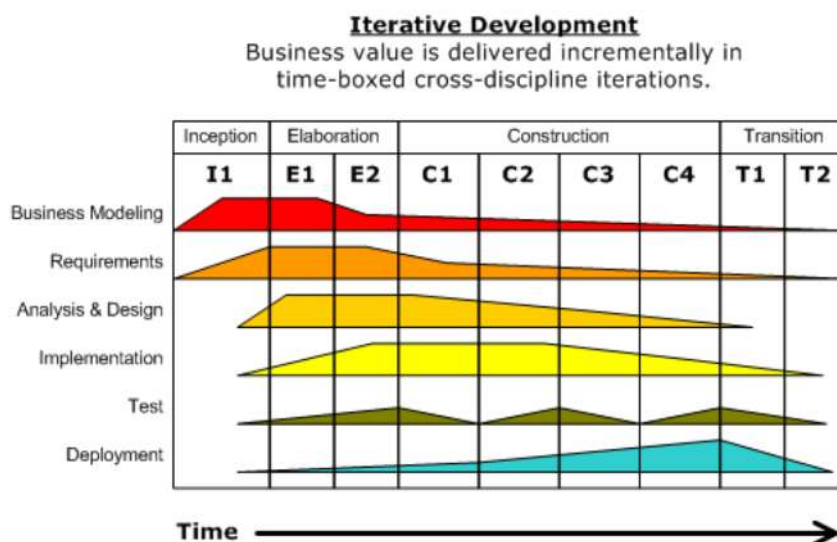
Unified process (UP).....	4
Grundlæggende principper.....	4
Iterationer.....	4
(Rational) Unified process.....	4
Modellering.....	4
Unified Modelling Language (UML).....	4
Analyse og Design (repeat).....	4
Inception fasen.....	4
Kravspecifikation.....	4
Krav.....	4
Use cases.....	6
Use case beskrivelser.....	6
Risikoanalyse.....	7
Risiko håndtering.....	7
Risiko analyse.....	8
Risiko strategier / planlægning.....	8
Domænemodeller.....	9
Hvad bruges domænemodellen til.....	10
Designklasse.....	10
Analyse og associationer.....	11
Løsning.....	11
Multiplicitet.....	11
Klassediagrammer.....	12
Generelt.....	12
Designklassediagrammer.....	13
Analyse vs designklasser.....	13
Design og implementering.....	13
Attributter.....	14
Associationer.....	14
Tommelfingerregel - Associationer.....	15
Dependency / afhængighed.....	16
Aggregation og composition.....	16
Aggregationer.....	17

Composition.....	17
Piletyper.....	18
Abstrakte klasser.....	19
Objekter og klasser.....	19
Design model - pakker.....	19
Sekvensdiagrammer.....	20
System-sekvensdiagrammer.....	20
Statisk og dynamisk modellering.....	21
Design sekvens diagram.....	22
Klasser i sekvens diagram.....	24
Aktivitetsdiagrammer.....	24
Handling (actions).....	25
Forgreninger og sammenfletninger.....	26
Løkker/iterationer.....	26
Forks og joins.....	27
Start og slutttilstand.....	28
Swim lanes.....	29
Guards.....	30
Oversigt.....	31
Generelt dokumentation af kode.....	31
Separation of concerns.....	32
Patterns.....	32
Design patterns og grasp.....	32
Grasp.....	32
Arv og polymorfi.....	33
Generalisering - arv.....	33
Review.....	34
Tilstandsdiagrammer.....	34
Undertilstande.....	35
Applikations-tilstand.....	35
Objekt-tilstand.....	35
Projektplanlægning.....	36
Plan.....	36
Work breakdown structure.....	36

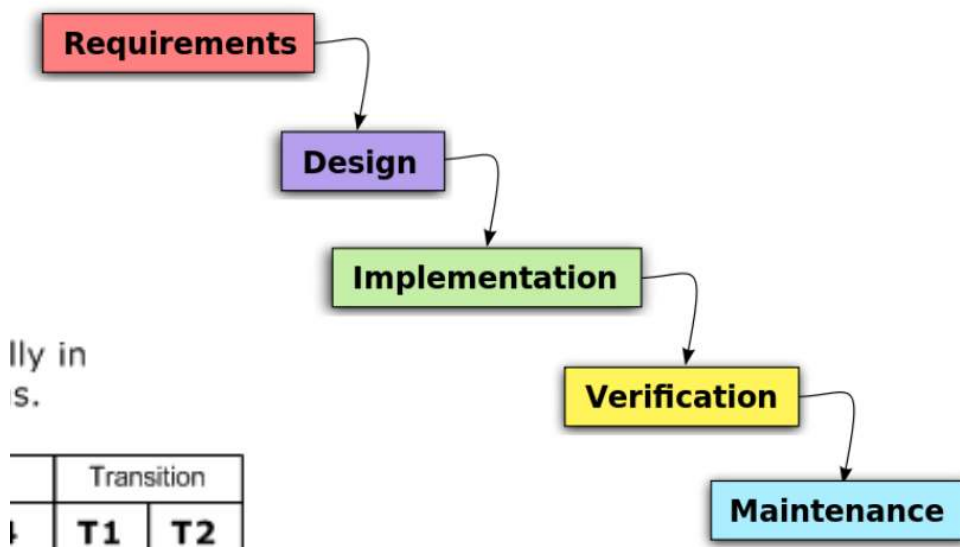
Organiser opgaver.....	36
Identificere kritisk vej.....	37
objekt kendskab / reference.....	37
Design elementer.....	37
Software kvalitet.....	38
Verificering og validering.....	38
Reviews and inspections.....	39

## Unified process (UP)

- To slags Agile og Waterfall
- Agile:



- Waterfall



## Grundlæggende principper

- Iterativ og inkrementel
- Use case drevet
- Risiko-drevet
- Arkitektur centreret

## Iterationer

- Opdelt i fase timeslots
  - Timeboxed (2-6 uger)
  - Delopgaver er fleksible
- Alle discipliner i hver iteration
- Hver iteration ender med fungerende program
  - Delelementer

## (Rational) Unified process

- 4 Faser
  - Projektets modenhed
- 6+3 discipliner
- 6 engineering
- 3 supporting
  - Configuration and change management
  - Project management
  - Environment

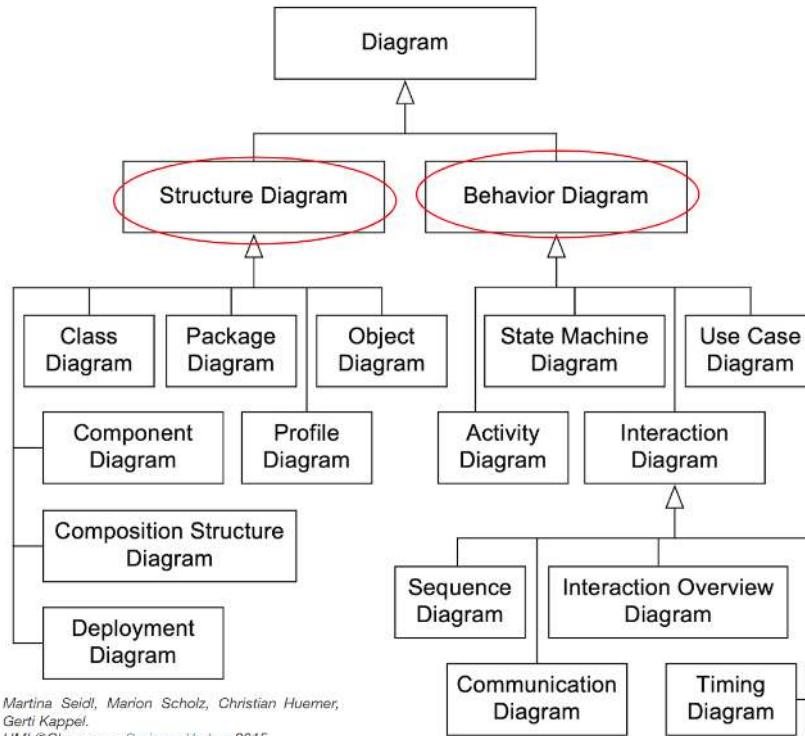
## Modellering

Systemet som model

- Forskellige modeller
  - Domæne
  - Test
  - Brug

- Diagrammer er indblik i modellen
  - Systemet fra et bestemt perspektiv
  - Systemet på et bestemt perspektiveringsniveau
    - Max 50 elementer på et diagram

## Unified Modelling Language (UML)



## Analyse og Design (repeat)

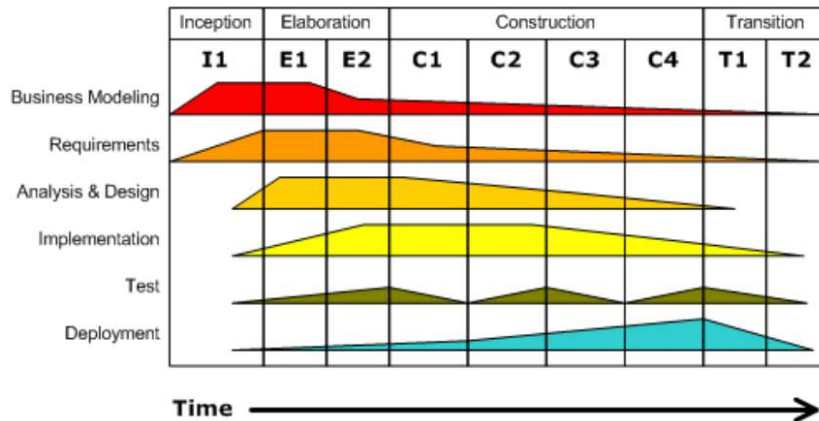
- Analyse
  - Undersøgelse af problem domæne og krav
  - Hvad skal dette software løse (Hvad?)
- Design
  - Konceptuel løsning
  - Hvordan skal dette software løse det (Hvordan?)

## Inception fasen

- Inception er ikke krav fasen
- Vis produkt scope, vision og business case
- Vision og business case
- Kan det gennemføres?
- Købes andetsteds eller skal det nyudvikles
- Skal arbejdet fortsættes?
- Go og no go

### **Iterative Development**

Business value is delivered incrementally in time-boxed cross-discipline iterations.



## Kravspecifikation

- Interessenter / stakeholders
  - Påvirker eller påvirkes af systemet
  - Har interesse i usecasen
- Aktører
  - Indgår i use cases
- Use cases
- Funktionelle krav
- Non-funktionelle krav

## Krav

- Funktionelle
  - Hvad systemet skal kunne (systemet validere pin-kode)
- Non-funktionelle
  - Hvordan det gør det (brugervenlighed, hastighed, implementering mfl.)
- FURPS+
  - Huskeregel
  - functionality, usability, reliability, performance, supportability
- Moscow
  - En måde at prioriterer på



# MoSCoW Analysis

FOR SETTING EFFECTIVE GOALS

M

## Must-Have

The absolute MUST. There is no way out and there is no shortcut.



S

## Should-Have

Essential but not vital



C

## Could-Have

Not a problem if it's left out but still is of significance.



W

## Will-Not-Have

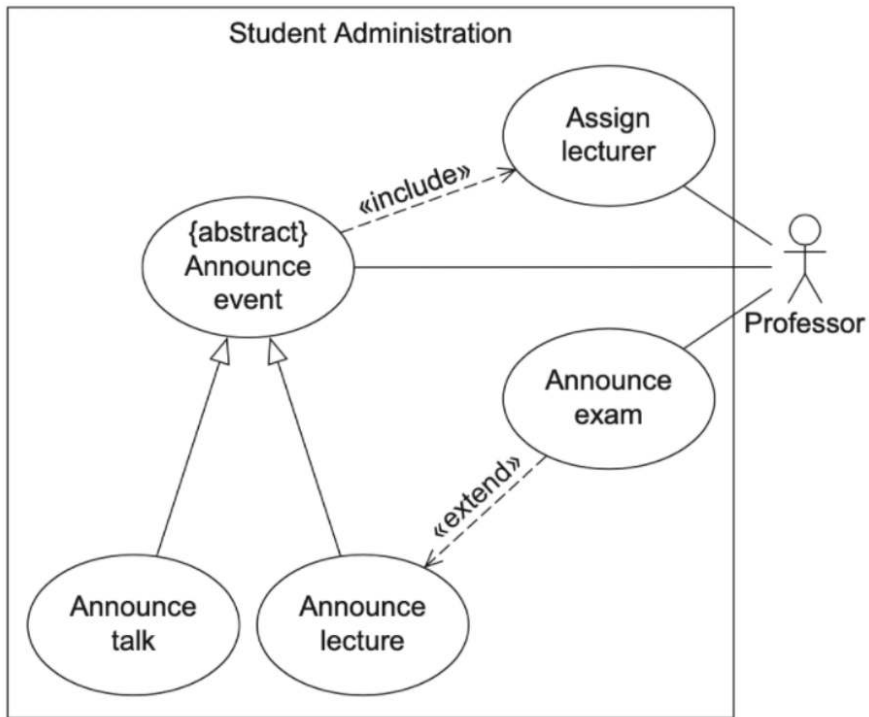
This is Irrelevant. Lose it. Not only for now, but for good.



○

## Use cases

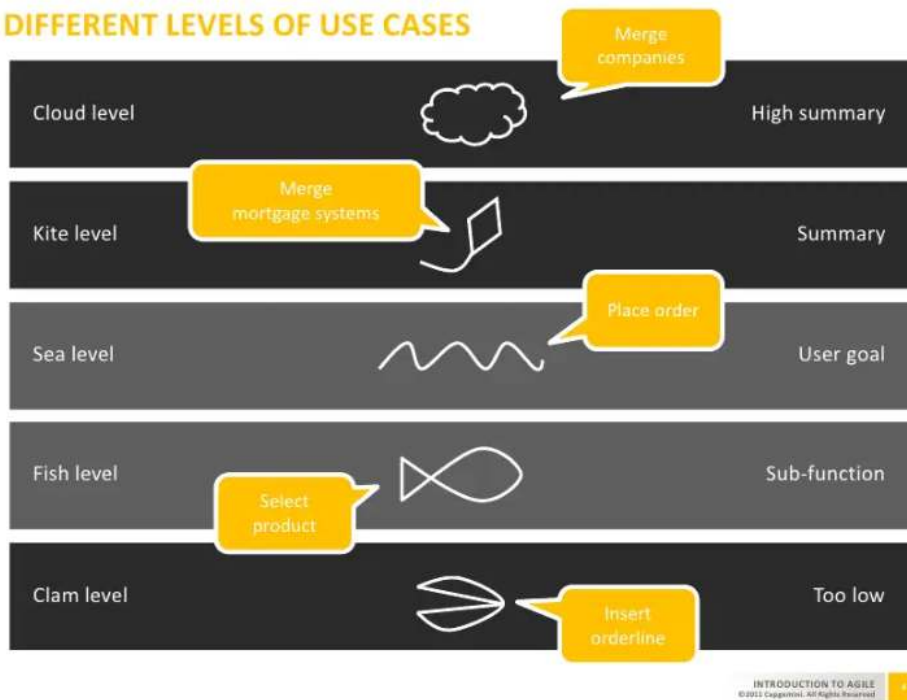
Et use case består af et verbum og et navneord, (reserver værelse), (tænd fjernsyn) osv.



## Use case beskrivelser

Forskellige detaljerings niveauer

## DIFFERENT LEVELS OF USE CASES



- Brief
  - Kort tekst om primært scenarie
- Casual
  - Flere afsnit evt. med alternative forløb
- Fully dressed
  - Alle detaljer, pre- og postconditions
- Aktører
- Preconditions
- Alternative flows, branches, løkker
- Postconditions

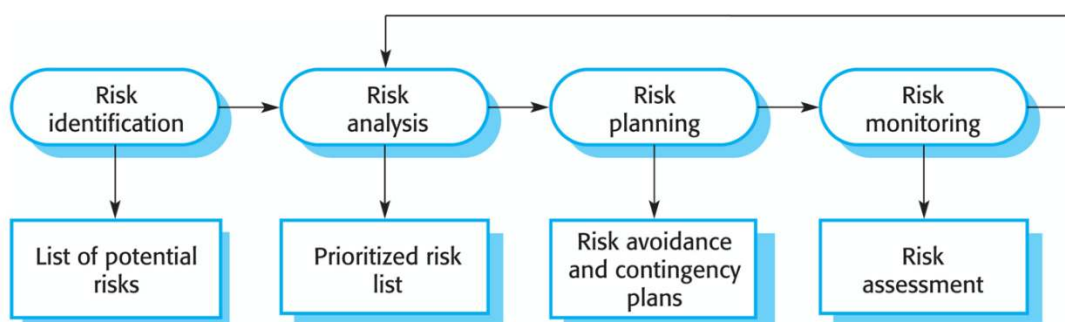
Use case: FindProduct
ID: 3
Brief description: The system finds some products based on Customer search criteria and displays them to the Customer.
Actors: Customer
Preconditions: None.
Main flow: <ol style="list-style-type: none"> <li>1. The use case starts when the Customer selects "find product".</li> <li>2. The system asks the Customer for search criteria.</li> <li>3. The Customer enters the requested criteria.</li> <li>4. The system searches for products that match the Customer's criteria.</li> <li>5. If the system finds some matching products then <ol style="list-style-type: none"> <li>5.1 For each product found <ol style="list-style-type: none"> <li>5.1.1. The system displays a thumbnail sketch of the product.</li> <li>5.1.2. The system displays a summary of the product details.</li> <li>5.1.3. The system displays the product price.</li> </ol> </li> </ol> </li> <li>6. Else <ol style="list-style-type: none"> <li>6.1. The system tells the Customer that no matching products could be found.</li> </ol> </li> </ol>
Postconditions: None.
Alternative flows: None.

- 
- Hvorfor use cases
- Fælles kommunikationsform
  - Nemmere for kunden at specificere og reviewe definition
  - Kunde deltagelse sikre færre fejl
  - Kunden kan selv skrive use cases
- Mindsker risikoen for at miste fokus
- Krav beskrives i en sammenhæng

## Risikoanalyse

### Risiko håndtering

- Identification
- Analysis
- Planning
- Monitoring



Copyright ©2016 Pearson Education, All Rights Reserved

## Risiko analyse (Risk rating)

- $P \times I$ 
  - Probability \* impact
- Sortering efter score
  - Grov prioritering
- Cutoff

### Risk Rating = Likelihood x Severity

Severity	Catastrophic	5	5	10	15	20	25
	Significant	4	4	8	12	16	20
	Moderate	3	3	6	9	12	15
	Low	2	2	4	6	8	10
	Negligible	1	1	2	3	4	5
			1	2	3	4	5
			Improbable	Remote	Occasional	Probable	Frequent
			Likelihood				

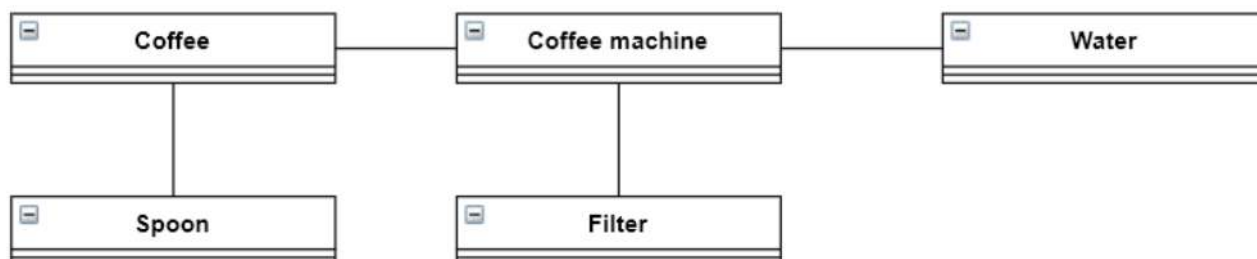
  

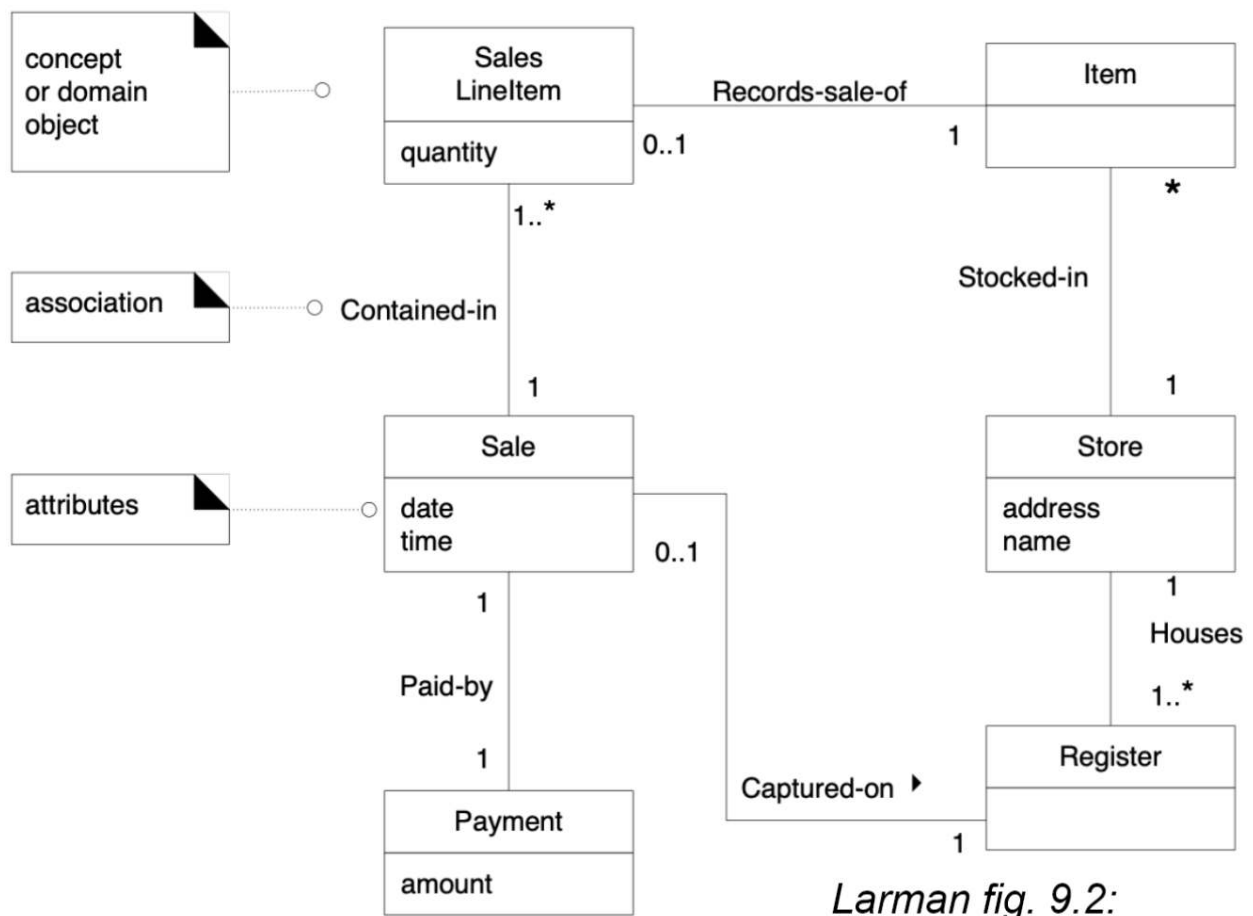
Catastrophic	STOP
Unacceptable	URGENT ACTION
Undesirable	ACTION
Acceptable	MONITOR
Desirable	NO ACTION

## Risiko strategier / planlægning

- Transfer
  - Videregiv ansvar
- Accept
  - Tag chancen
- Mitigate
  - Minimer konsekvenserne
- Eliminate
  - Fjern problemet

## Domænemodeller





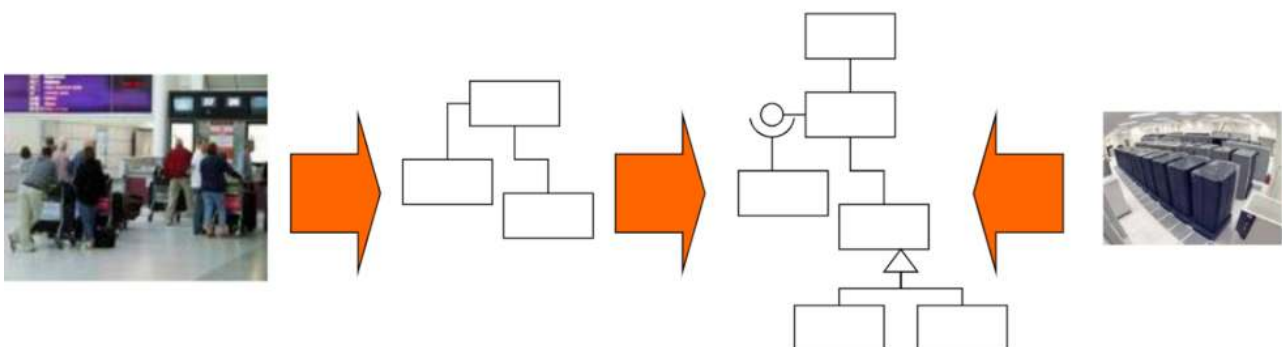
Hvad bruges domænemodellen til

Problem domain

Analysis classes

Design classes

Solution domain

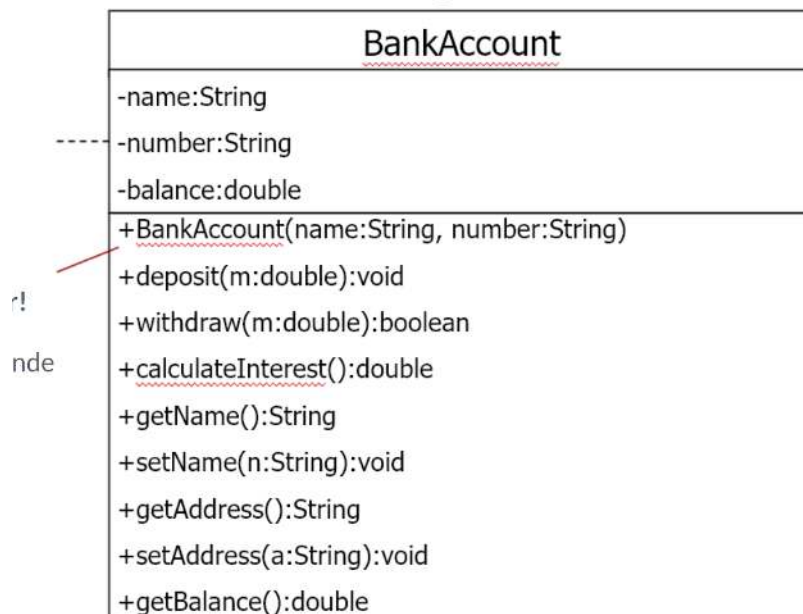


## Designklasse

- Alle attributter
  - (Der skal implementeres)
- Typer

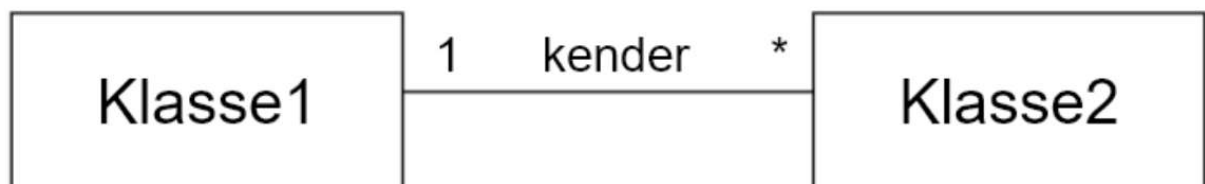
- Int, String eks
- Synlighed
  - + public
  - - private
- Alle metoder
- Ingen getters og setter

design



## Analyse og associationer

- Association
  - Forbindelser mellem klasser
  - Relationer mellem objekter af klassen type
- Angives
  - Linje
  - Navn
  - Multiplicitet
  - Evt retning

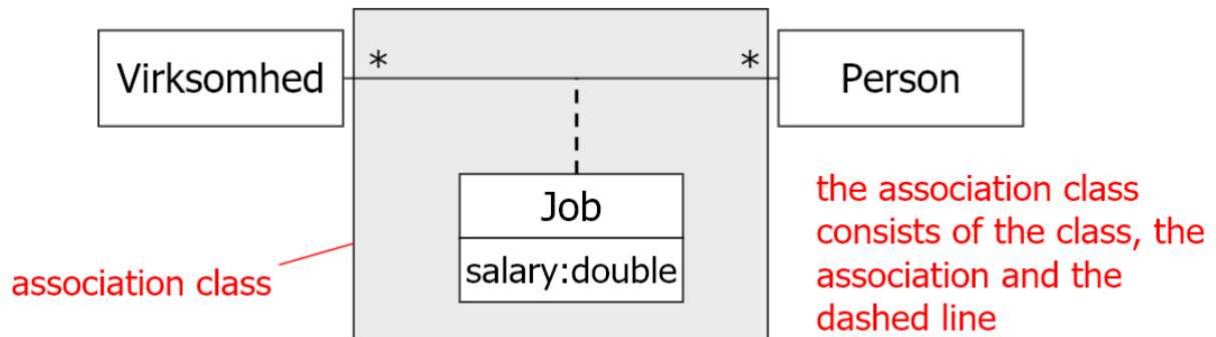


## Løsning

Associationsklasse

Instans for hver relation

Markeres med stiplet linje



## Multiplicitet

Vigtigt i klassediagrammer - Særligt for dataklasser

- Eks
  - Et firma kan have mange ansatte (employer)
  - En person er ansat i netop et firma (ingen bijobs eller arbejdsløshed)



multiplicity syntax: minimum..maximum	
0..1	zero or 1
1	exactly 1
0..*	zero or more
*	zero or more
1..*	1 or more
1..6	1 to 6

## Klassediagrammer

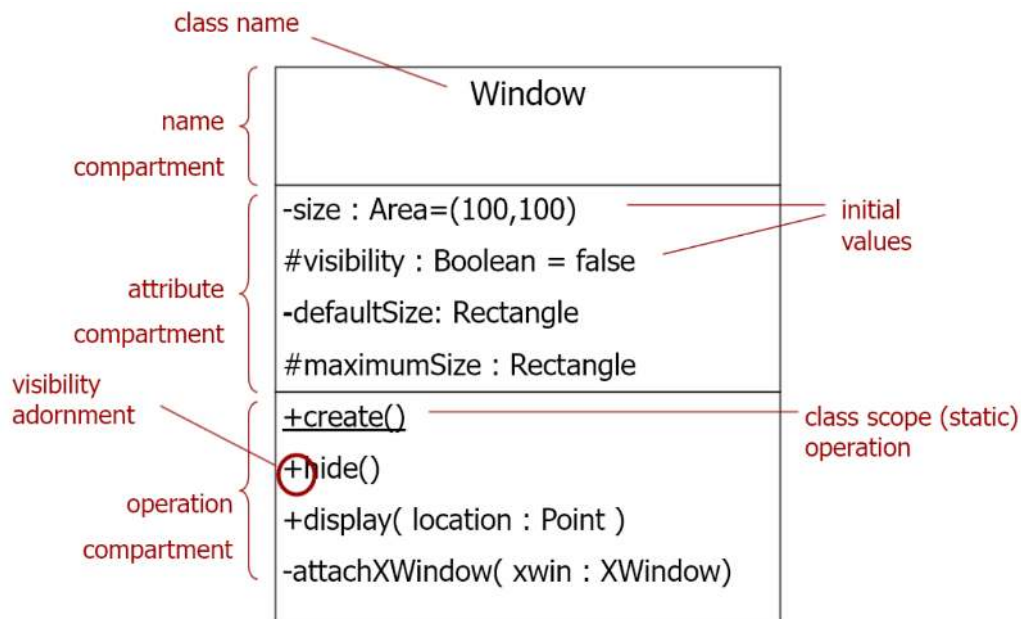
### Generelt

- Kan bruges til at beskrive domænet og designet
- Klassediagrammer benyttes til struktur/ statisk modellering
  - Objekter
- Analyse

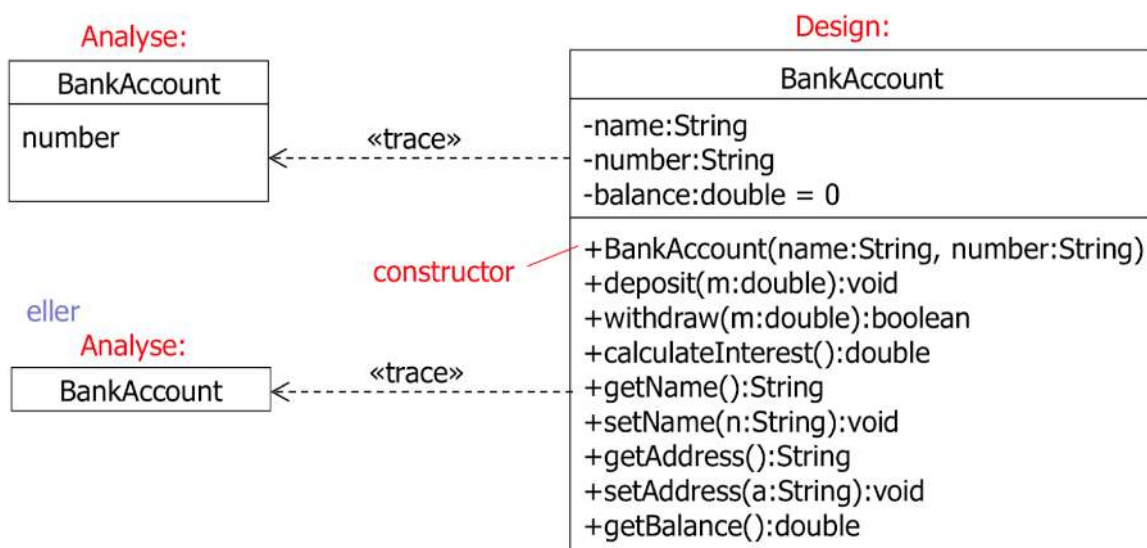


- Begrebsmodel / domænemodel
- Klassenavn og evt. attributter
- Design
  - Softwaremodel / designmodel
  - Klassenavn, attributter og operationer
- En designmodel kan indeholde 10 til 100 gange flere klasser end analysemodellen
  - Klasser der ikke finde i domænet
    - Klasse til designmønstre
    - Controlleres, adaptors

## Designklassediagrammer



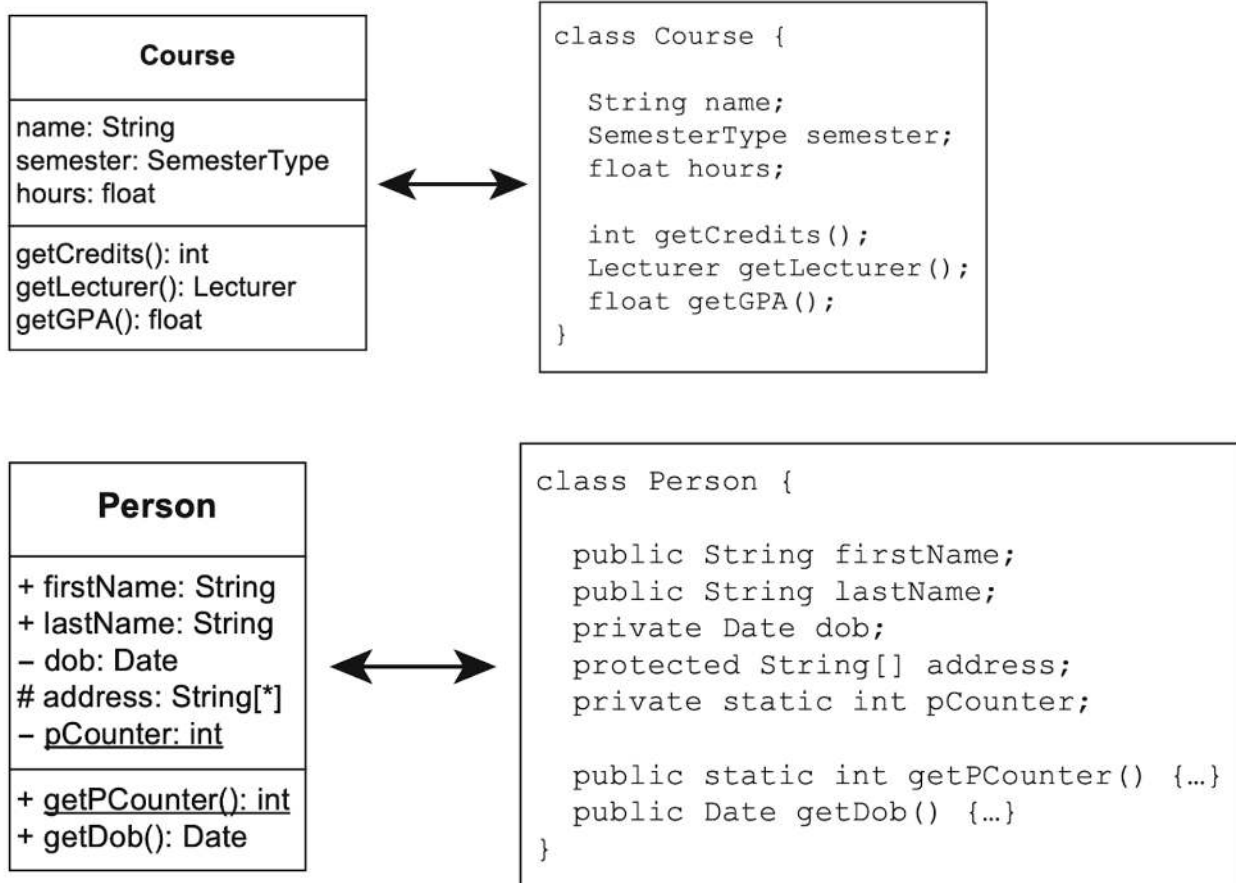
## Analyse vs designklasser



*Arlow & Neustadt*

## Design og implementering

- Klasse definition i UML vs Java
  - Syntaks forskelle

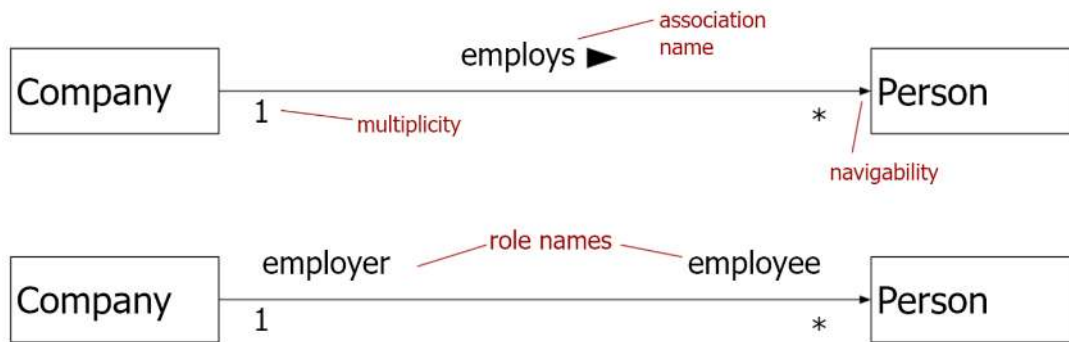


## Attributter

- **Visibility**, **Navn**, **Type**, evt. **multiplicitet**, evt. **default værdi**, evt. **property**
  - **visibility** **name** : **type** **multiplicity** = **default** {property string}
- Eks:
  - - passengerList : String [0..\*] = checkedInList {ordered}
  - - age : int
  - + GRAVITY : double = 9.81 {readOnly}

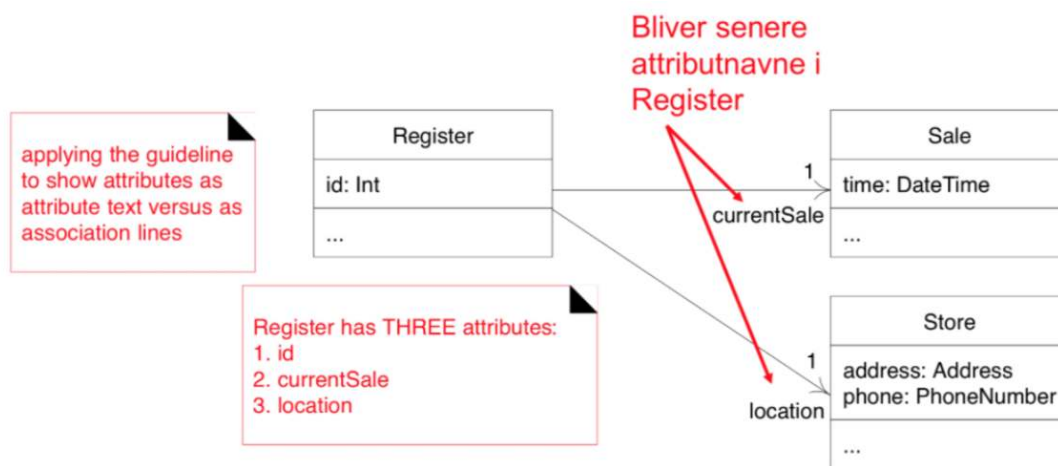
## Associationer

- Forbindelse mellem klasser
- Angiver en form for forbindelse mellem klasserne
  - (bruges typisk til at sende meddelelser (metodekald) mellem klasserne)
- Repræsenterer en reference til et objekt



## Tommelfingerregel - Associationer

- Simple typer: Attribut
- Klasser: Associationer (eller attributter)



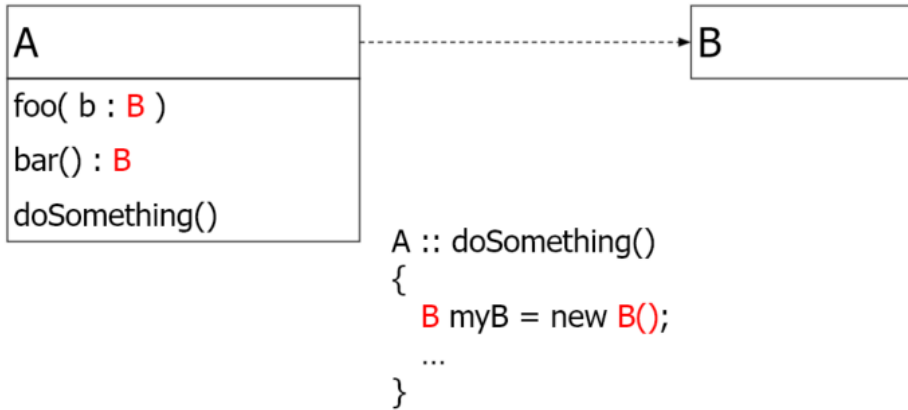
Brug:

- attributnavn ved simple datatyper
- associeringslinie til klasser

Larman fig. 16.5

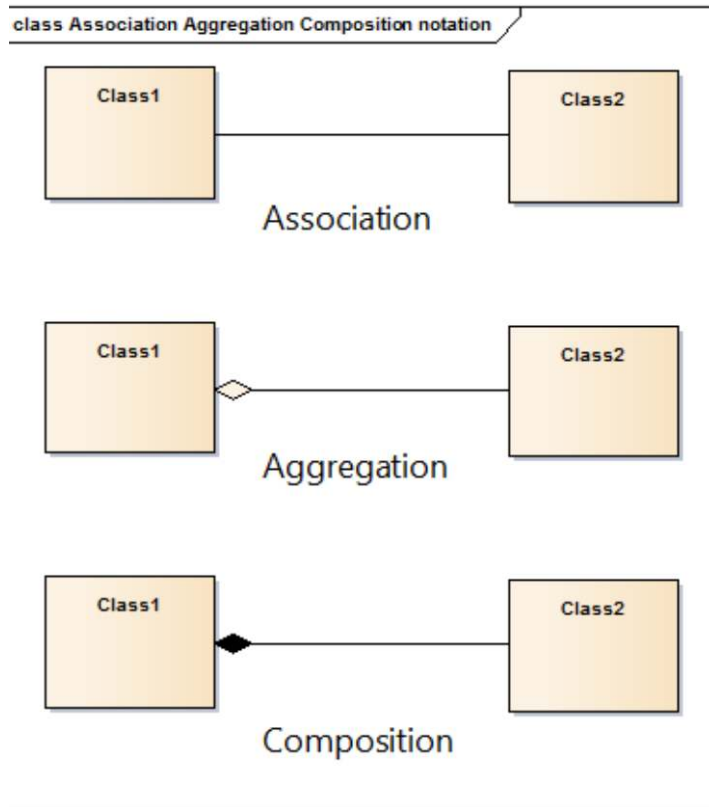
## Dependency / afhængighed

- "Light-association"
- Midlertidig reference
- Ændring i en klasse påvirker en anden



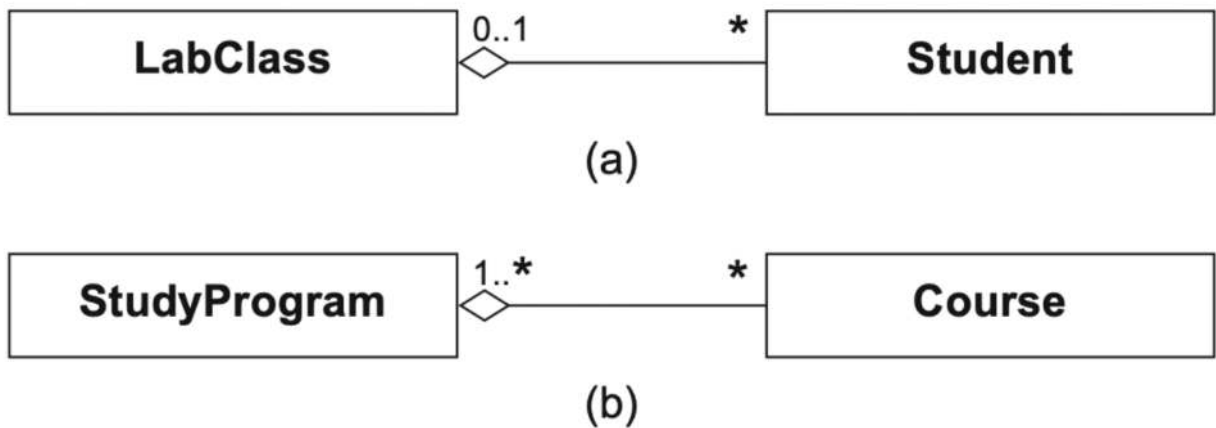
## Aggregation og composition

- Modellere "helhed - del" forhold
- To former:
  - Aggregering (delt/"shared")
  - Komposition
- Aggregering - "Dele" eksistere uafhængigt af "helen"
  - Eks: printere og computere
- Komposition er en stærkere form for aggregering - "dele" eksistere ikke udenfor "helheden"
  - Eks: Hus af mursten
- Sjældent nødvendigt at skelne i java
- Pile i klasse associationer



## Aggregationer

- LabClass kan have studerende
- StudyProgram kan have Courses



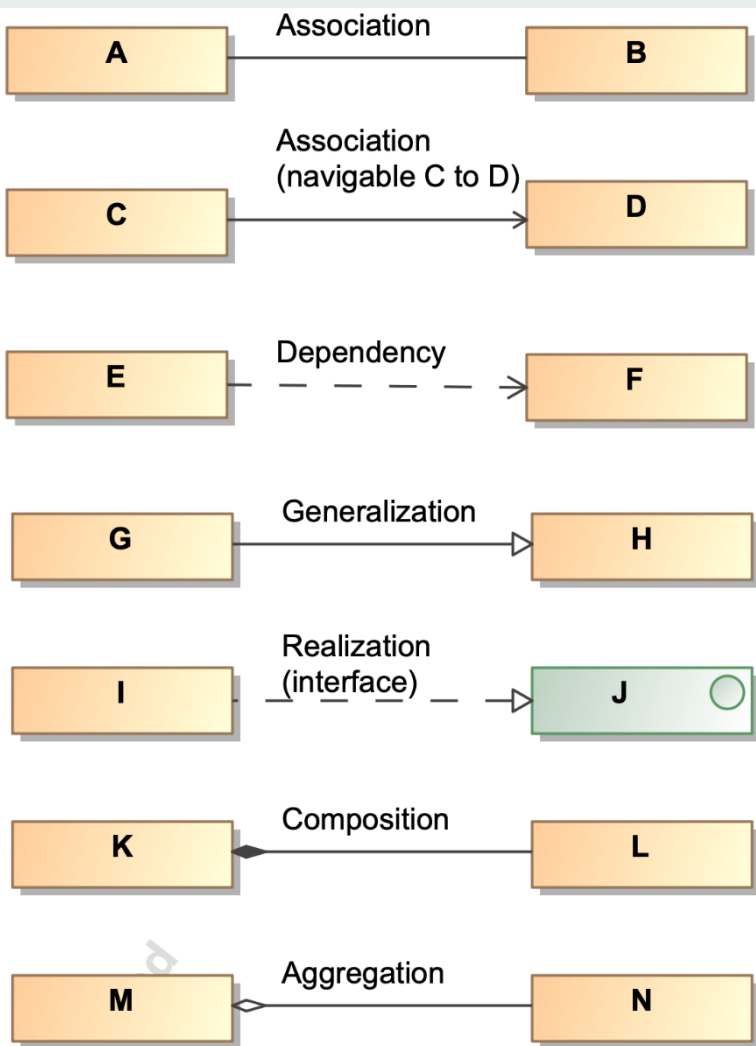
## Composition

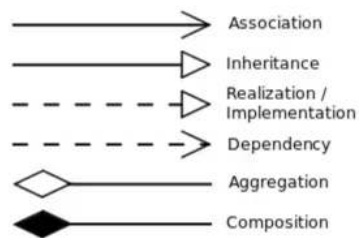
Stærk kobling

Under objektet tilhører 0 til 1 stamobjekt



## Piletyper





- **Association relationship:** is a general one.
- **Inheritance relationship:** connects a *subclass* with its own parent class (see OOP inheritance paradigm) from which inherits all attributes & methods.
- **Realization relationship:** connects a class with the *interface* which implements (see OOP interface implementation).
- **Dependency relationship:** connects a class with another one that is strictly needed by the first in order to achieve its tasks.
- **Aggregation relationship:** connects a class with another one that's a part of it due to the software / database structure previously projected. *For example in diagrams of an OOP class, a variable could be not a general String or Integer but a particular class. In this case the class you are defining has an "aggregation relationship" with the class of that variable.*
- **Composition relationship:** connects a class with another one that — physically in the real world — is part of it.

## Abstrakte klasser

Kan ikke instantieres

**Person**

**{abstract}  
Person**

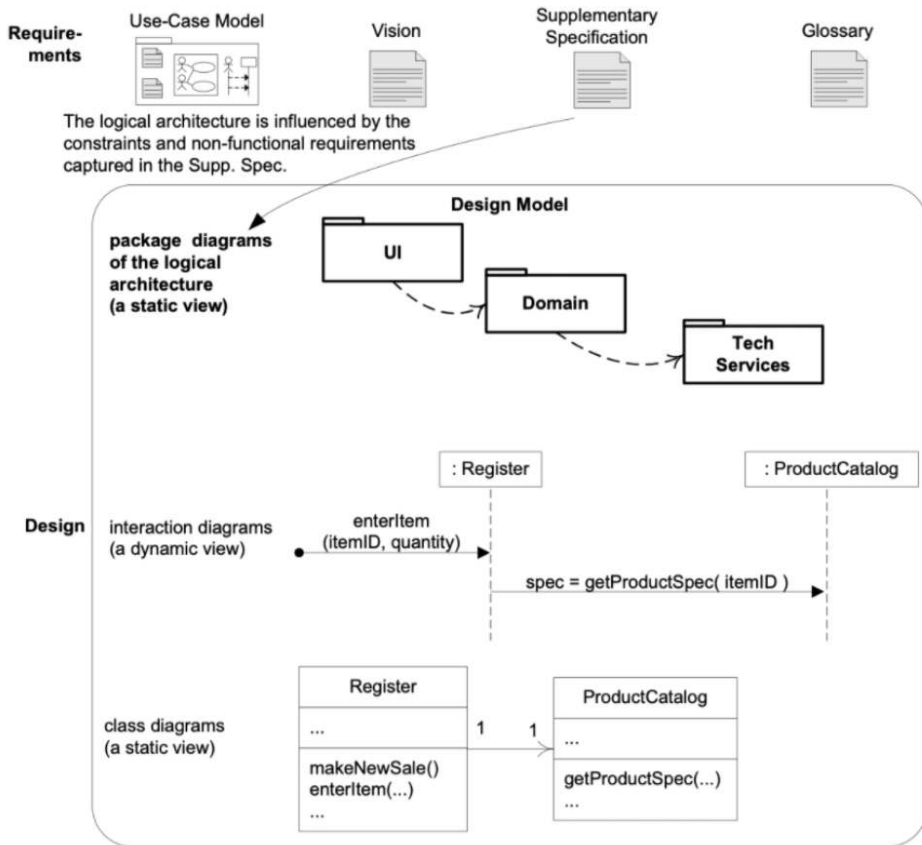
## Objekter og klasser

- Klasse: arbejdstegning
- Objekt: Et konkret forekomst/Instans af en klasse
- Atributter(+Associationer) /Felter
  - (Variable på klasser/objekter)
- Operationer/metoder
  - (funktioner på klasser/objekter)
- Design
  - Signaturer besluttet

## Design model - pakker

- Logisk gruppering af elementer
  - F.eks. klasser

- Kan vise logisk arkitektur af systemet
- Ræpresentere et "namespace"
- Kan være nested
- Eks - logisk segmentering af program



## Sekvensdiagrammer

- Modellere et specifikt scenarie
  - Use case
- Ofte kun en del af handlingen

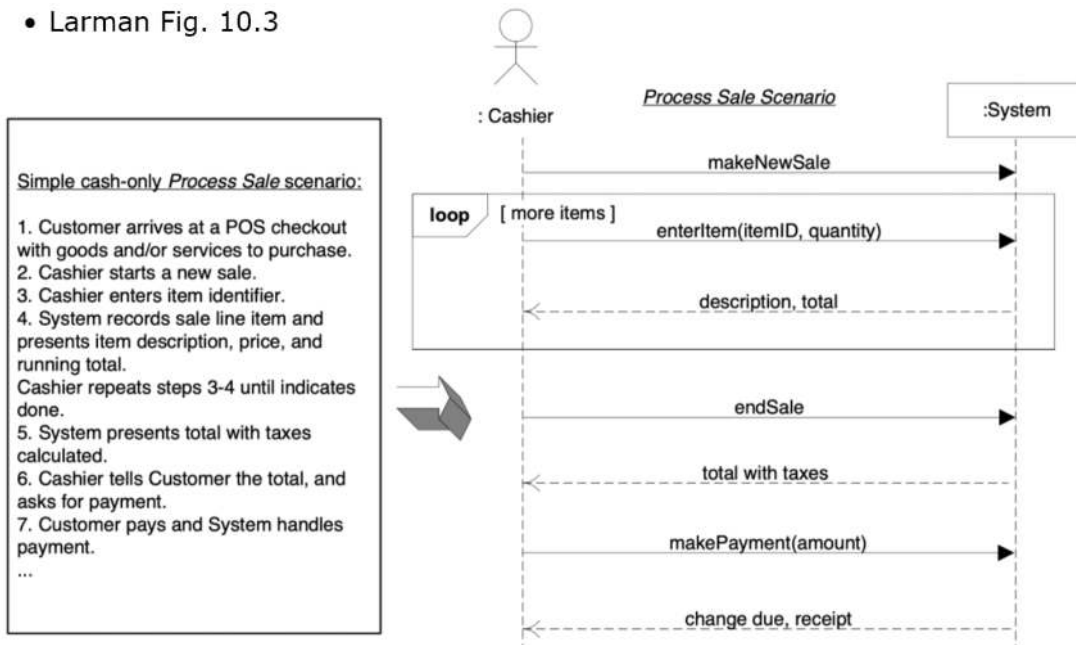
## System-sekvensdiagrammer

Interaktion mellem

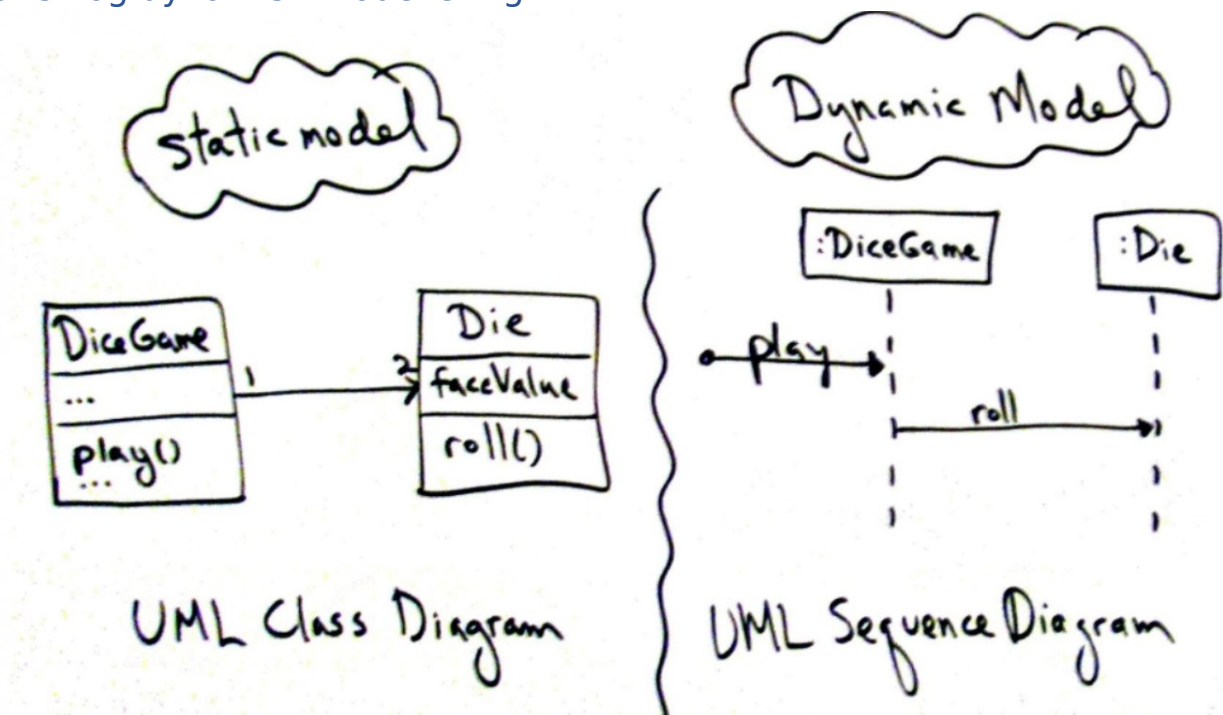
- Visualisere en brugers interaktion med et system
- Aktør
- System
- Visuel repræsentation af use cases



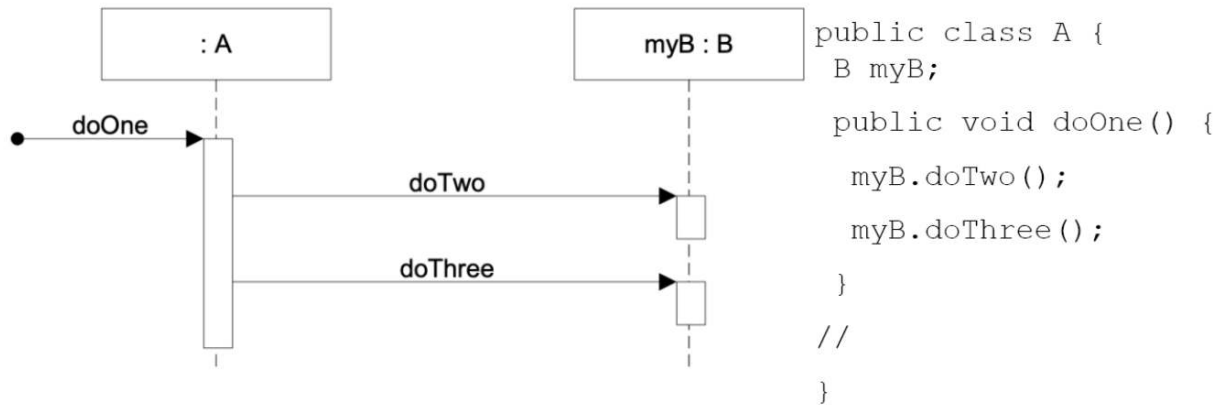
• Larman Fig. 10.3

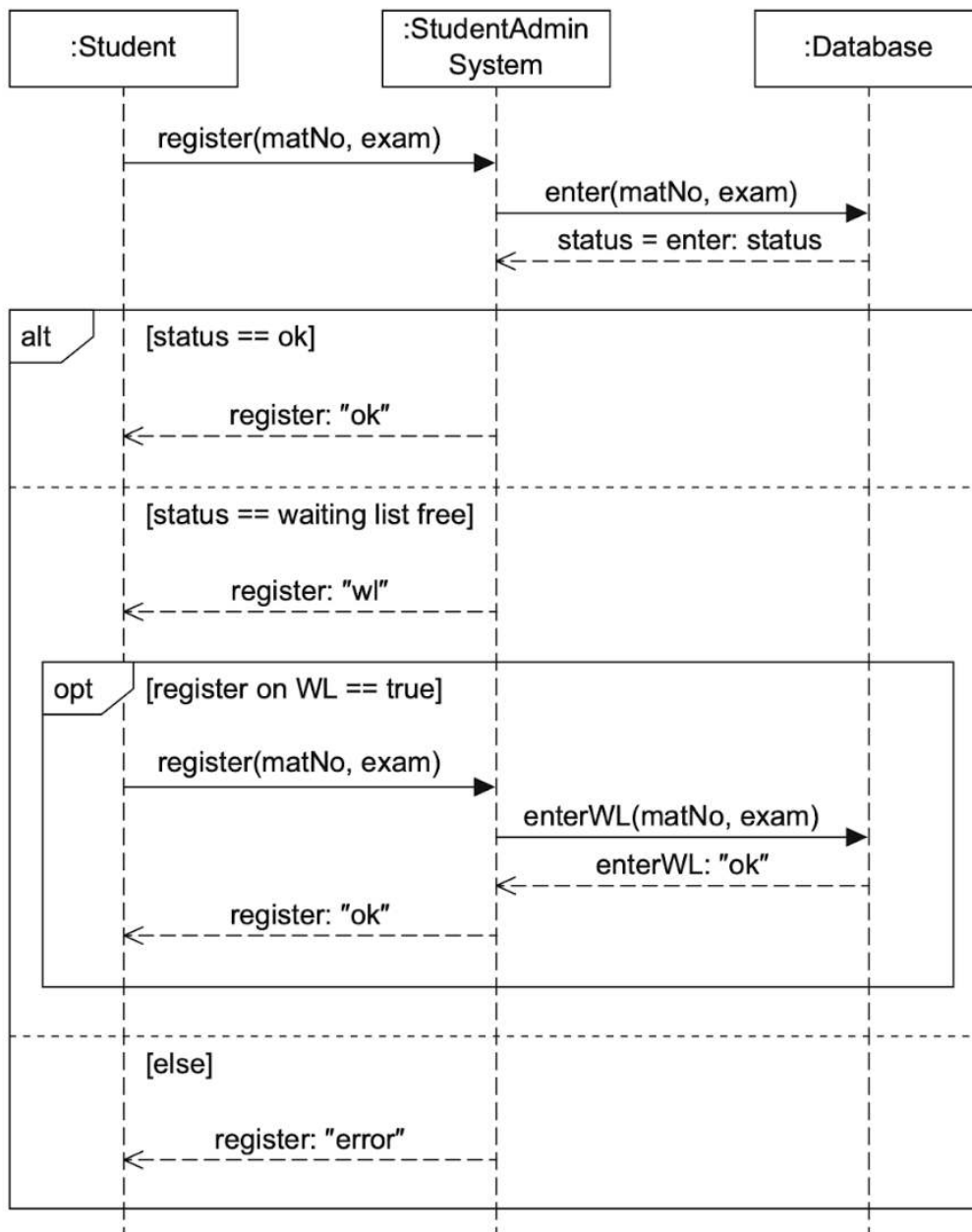


Statisk og dynamisk modellering

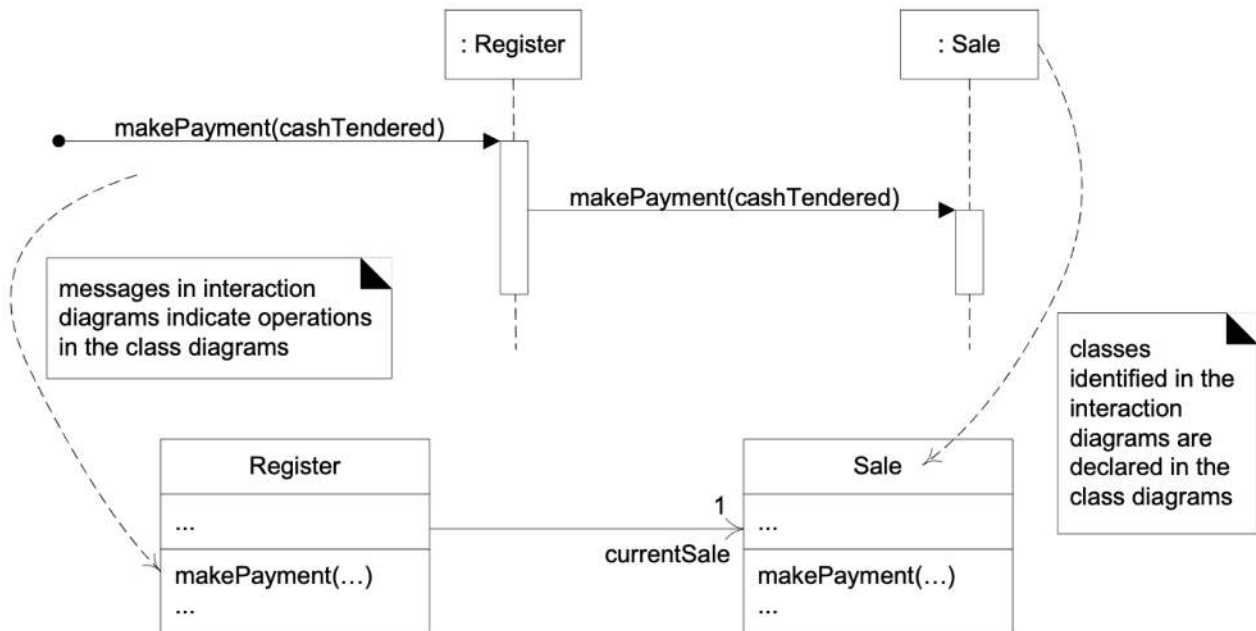


## Design sekvens diagram



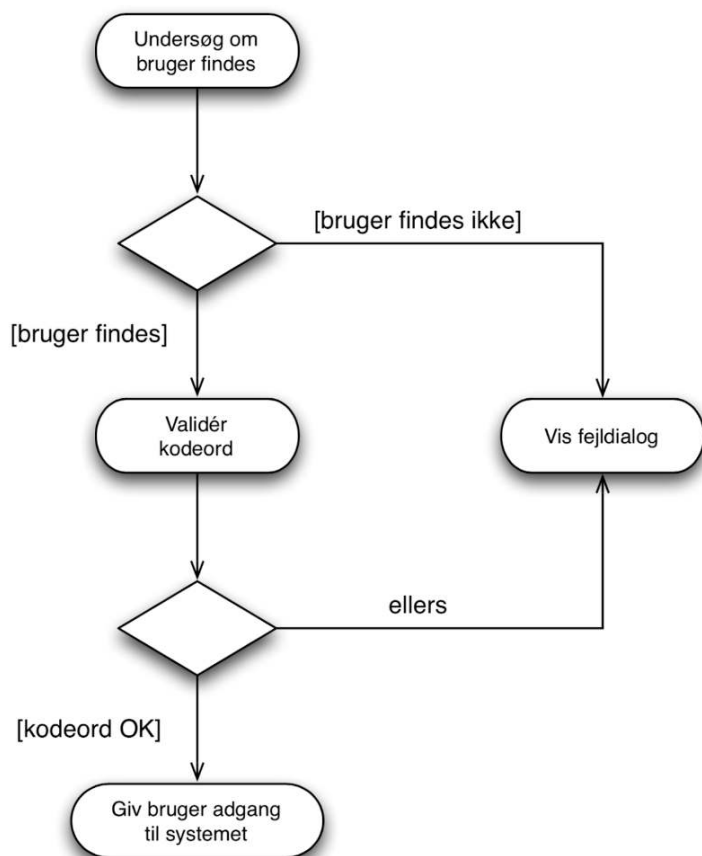


## Klasser i sekvens diagram



## Aktivitetsdiagrammer

- Aktivitets-diagrammer er dynamiske er dynamiske diagrammer
- UML-Udgaven af et flow-chart
- Dynamisk diagram, der viser et flow i en proces, algoritme
- Aktivitet
  - Noget et objekt udfører
  - Ikke atomar
  - Kan opdeles i et antal simple atomare handlinger (actions)
  - Har som regel en beskrivelse



## Handling (actions)

- Trin i aktiviteten
- Operation der kan udføres af objektet på andet objekt
  - Eller en simpel beregning
- Atomar - kan ikke opdeles i simplere handlinger
- En aktivitet forventes at tage et stykke tid
- En handling bruger meget kort tid
- 

FID = findes(ID)

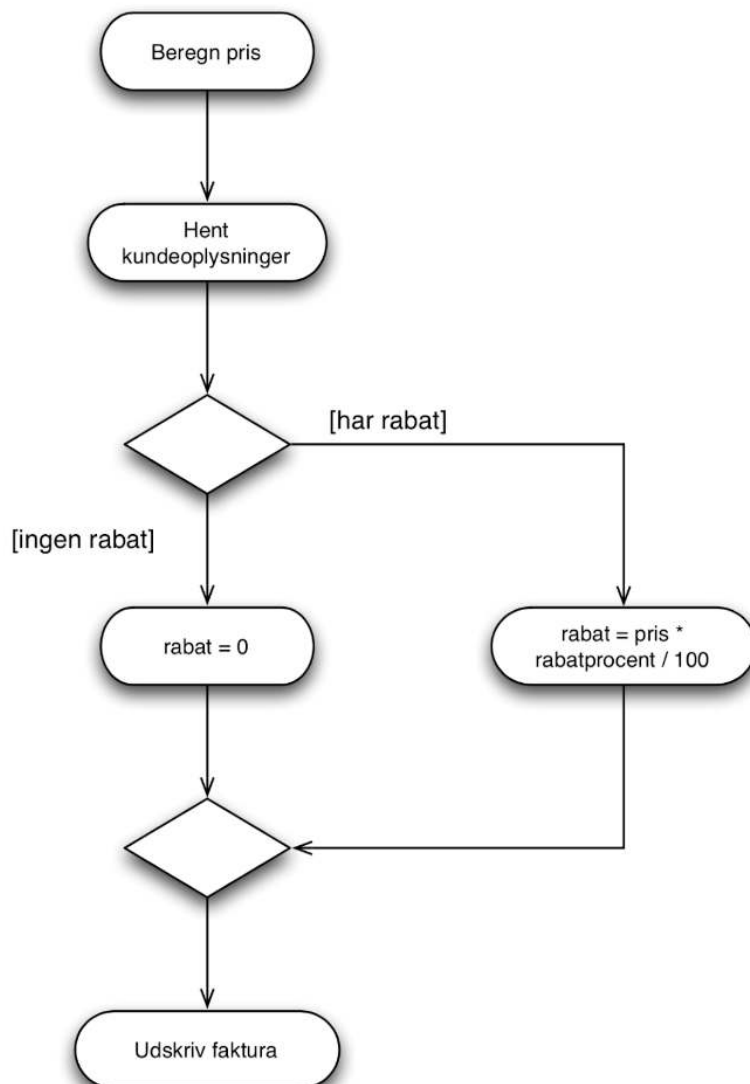
visBruger()

saldo = saldo +  
beløb

saldo = saldo +  
k1.hæv(beløb)

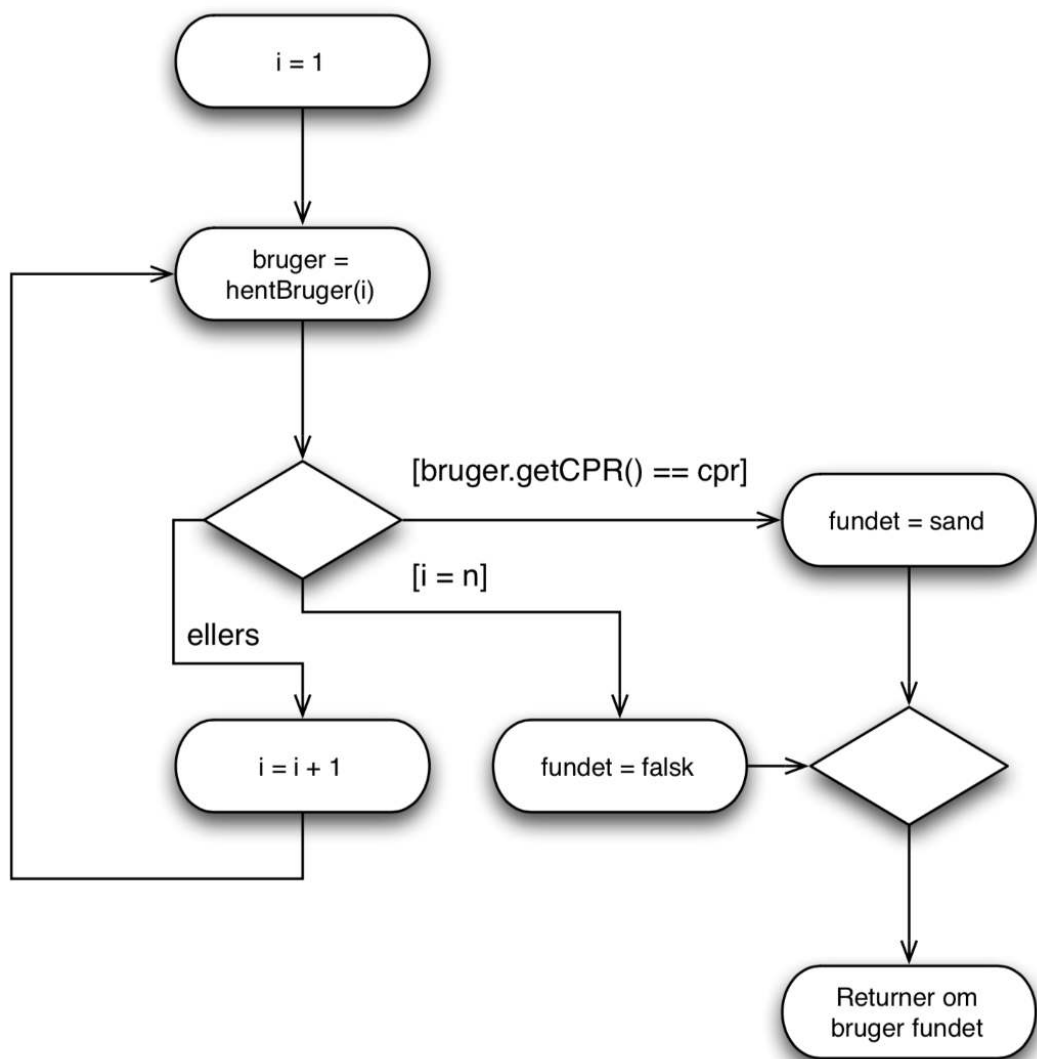
k2.indsæt(k1.hæv(beløb))

## Forgreninger og sammenfletninger



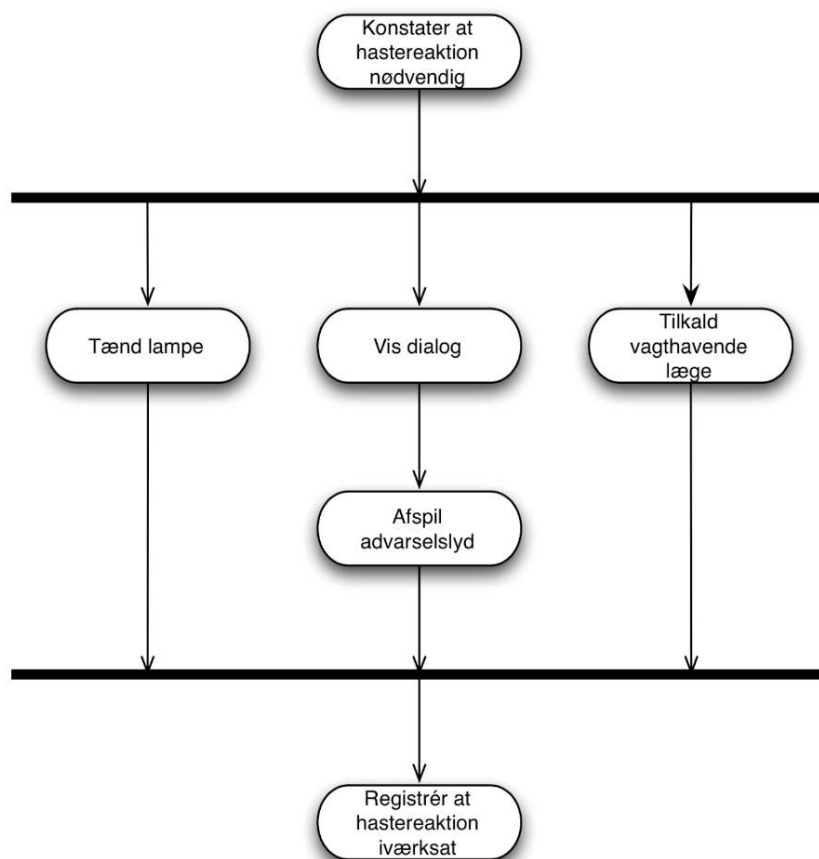
## Løkker/iterationer

Sammenfletning over forgrening



## Forks og joins

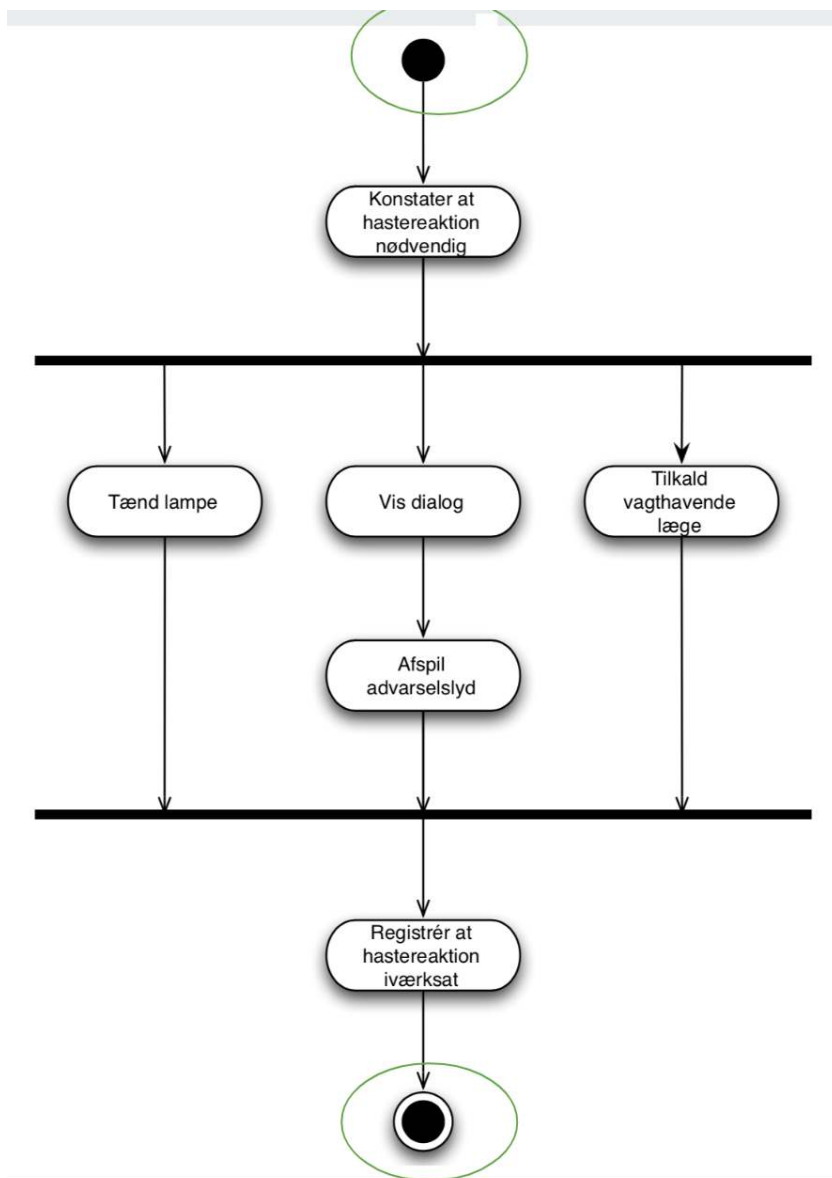
- Flere ting foregår parallelt
  - Flertrådet applikation



## Start og sluttilstand

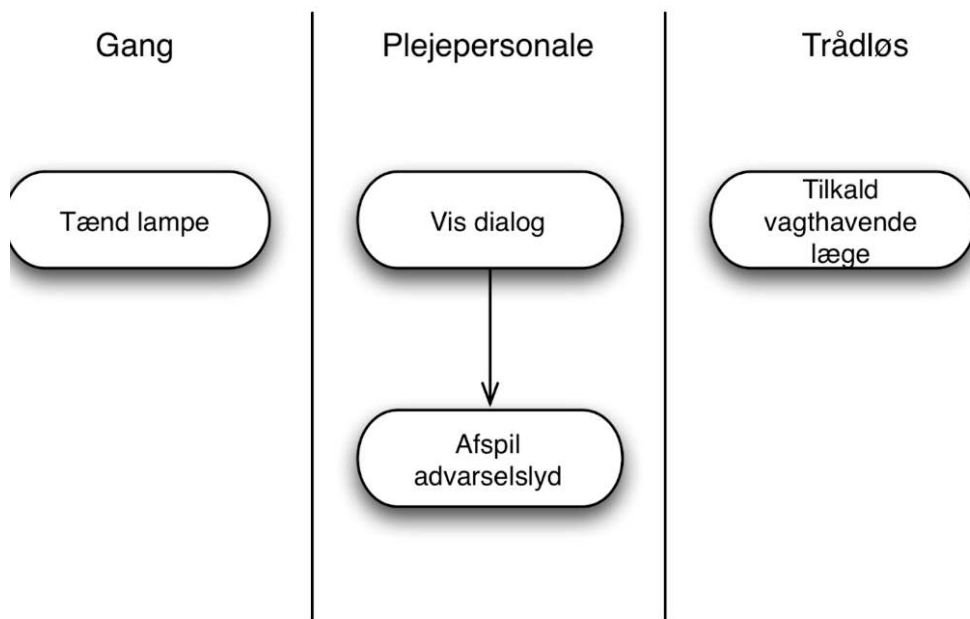
Der kan være mere end en





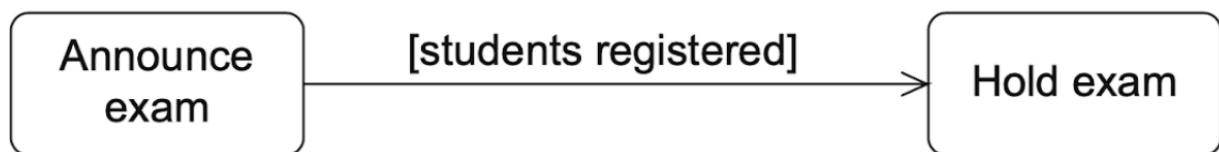
## Swim lanes

- Anvendes hvis der er flere adskilte områder/systemer der samarbejder om en aktivitet
- Adskilles af lodrette streger






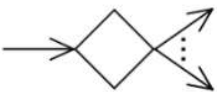
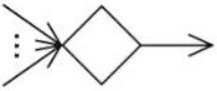
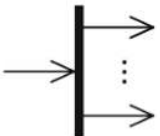
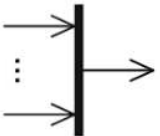
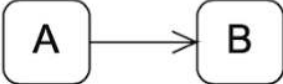


## Guards

Boolske udtryk - kan besvare med ja/nej



## Oversigt

<i>Name</i>	<i>Notation</i>	<i>Description</i>
Action node		Actions are atomic, i.e., they cannot be broken down further
Activity node		Activities can be broken down further
Initial node		Start of the execution of an activity
Activity final node		End of ALL execution paths of an activity
Flow final node		End of ONE execution path of an activity
Decision node		Splitting of one execution path into two or more alternative execution paths
Merge node		Merging of two or more alternative execution paths into one execution path
Parallelization node		Splitting of one execution path into two or more concurrent execution paths
Synchronization node		Merging of two or more concurrent execution paths into one execution path
Edge		Connection between the nodes of an activity

## Generelt dokumentation af kode

- Selvdokumenterende elementer
  - Klassenavne - Svarer til ansvarsområde
  - Metodenavne - svarer til funktionalitet
  - Variabelnavne - svarer til indeholdet
- Layout

- Opdeling - eks
  - Ekstra linjer mellem metoder
  - Separat fil til klasser (som regel)
- Indrykning
  - Skift af scope
- Kommentarer
  - Kun når koden ikke kan læses umiddelbart
    - Algoritmiske løsninger

## Separation of concerns

- Hyppigt forekommende designprincipper
- Opdeling af et system i separate områder
- Modularitet i koden
  - Opdeling af udvikling
  - Læsbarhed
  - Nemmere isolering af fejl
  - Nemmere tests
  - Genbrug
  - Nemmere udskiftning

## Patterns

### Design patterns og grasp

- Design pattern
  - Design løsning til ofte forekommende problemer
  - Generaliserbar
- GRASP
  - General Responsibility Assignemt Software Patterns
  - Klasser har ansvar
- GoF
  - Gang of Four
  - Endnu et sæt af software patterns

### Grasp

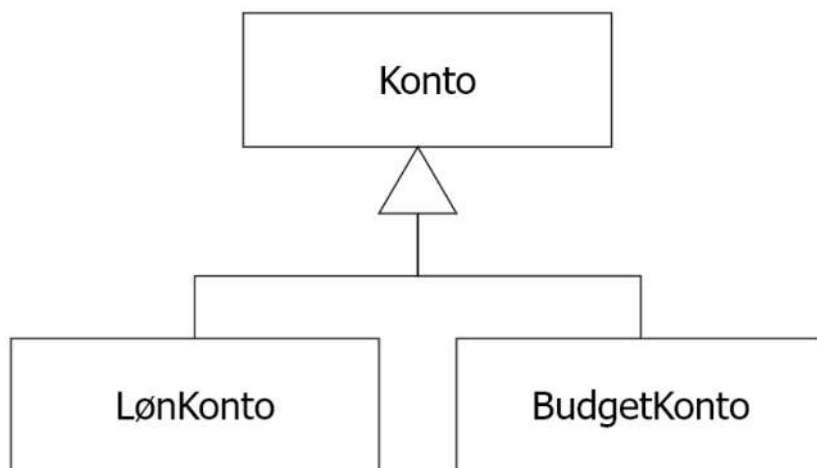
- Creator
  - Hvem er ansvarlig for oprettelse af klassen?
  - B er creator for A hvis
  - B aggregerer A
  - B indeholder A
  - B benytter A
  - B har initialiseret data for A
- Information Expert
  - Princip for tildeling af ansvar til objekter
  - Klassen har nødvendig information til at opfylde ansvaret
- Low coupling
  - Lav kobling

- En klasse skal være så uafhængig som muligt
- En klasse skal kun associeres med de få klasser, der er nødvendige for at den kan opfylde sit ansvarsområde
- High cohesion
  - Høj sammenhørighed
  - En klasse skal have et veldefineret ansvarsområde
  - En klasse skal have et sæt operationer, der understøtter ansvarsområdet
- Controller
  - Tildel controller ansvar til
    - System klasse
    - Klasse som repræsenterer usecase scenarie
  - Pas på med alt for store controllers
- Polymorphism
  - Når flere typer af objekter kan noget ensartet
- (pure fabrication)
  - Introduktion af en klasse der ikke findes i problemdomænet
  - For at opnår low coupling og high cohesion
  - GoF Design patterns er pure fabrication
- (Indirection)
  - For at introducere lav kobling mellem objekter
  - Eks database-adaptor
    - Gør det potentielt lettere at skifte database
    - Skaber evt lettere interaktion
  - Eks terningbæger
    - Både pure fabrication og indirection
- (Protected variations)
  - Beskyt objekter mod ændringer i andre objekter
  - Hvis der er noget som kan ændre sig
  - Introducer stabil grænseflade

## Arv og polymorfi

### Generalisering - arv

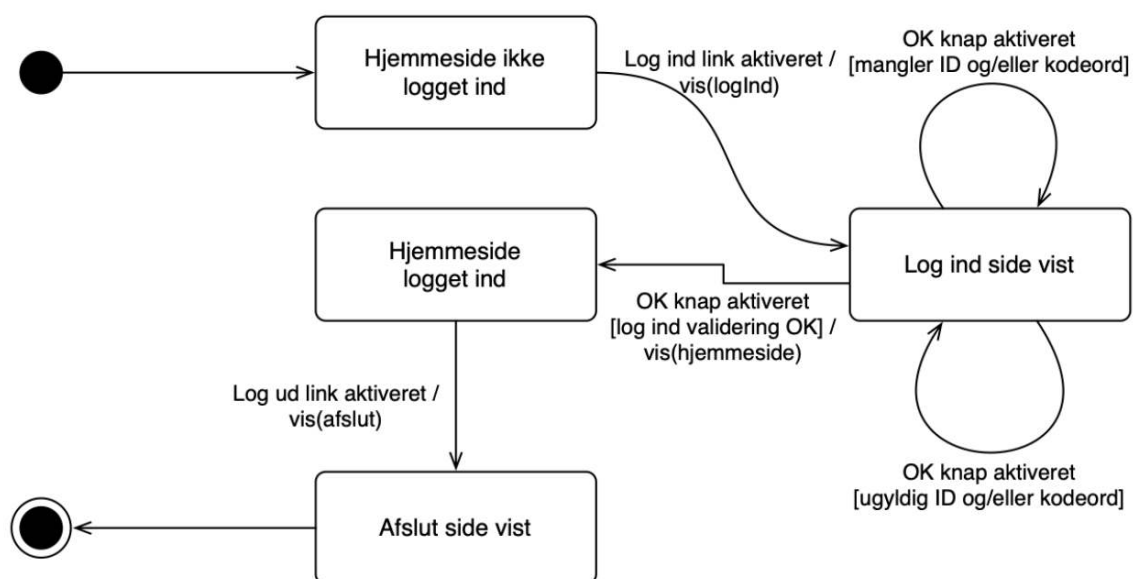
- Egenskaber fra superklasse nedarves til subklasser
- Subklasser kan tilføje egen specialisering



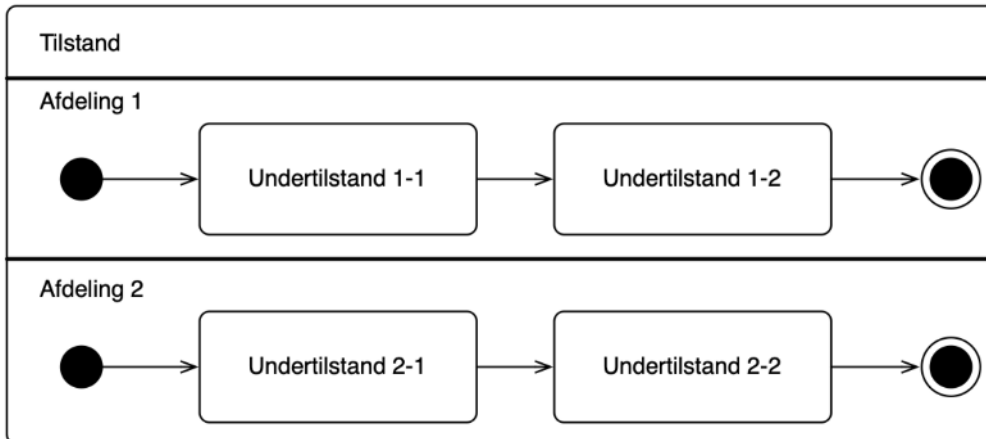
•

## Review

## Tilstandsdiagrammer



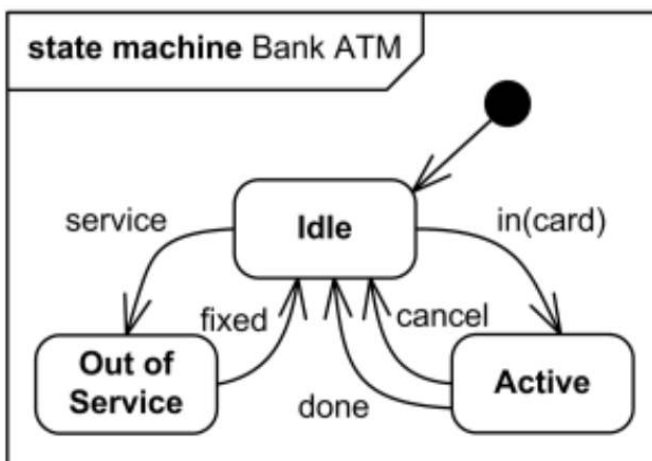
## Undertilstande



## Applikations-tilstand

En applikation har altid en tilstand

Applikationens tilstand udgøres af objekternes tilstand



## Objekt-tilstand

- Et objekt kan være hvad som helst
  - Bruger
  - Bil
  - Microbølgeovn
  - Applikation
    - Eks. Navigation, Login Tilstand
  - Odre
  - Aflevering
  - Eksamen
- Whitebox: Tilstanden udgøres af
  - Objektets attributters tilstande
- Tekstuel
  - Fil åben
  - Knap nedtrykket

- Logget ind - Logget ud

## Projektplanlægning

- Project scope statement
  - Formål, ønsket resultat, acceptance criteria, udeladt
  - Retning
- Project plan
  - Delopgaver, estimer, rækkefølge, deadlines
  - Rute

## Plan

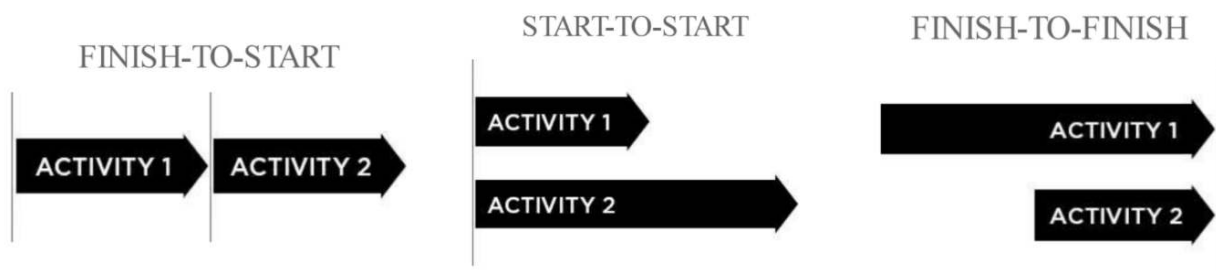
- Risk management
- Project schedule
- Budget
- Communication plan

## Work breakdown structure

- Nedbryd projektet i overskuelige bidder
  - Hvis bidden er for svær at estimere, er den måske for stor
  - Meget store elefanter skal deles ud for ikke at blive rådne
- Mindmap
- Post its
  - Kanban
  - Scrum
- Liste

## Organiser opgaver

- Mindmaps og post its konverteres til liste
- Bindinger (Dependencies) identificeres
  - Finish to start
  - Start to Start
  - Finish to finish
- Opgaver, der er bundne sættes efter hinanden
  - Bindinger er vigtige for kritisk vej



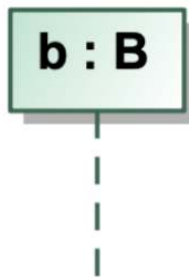
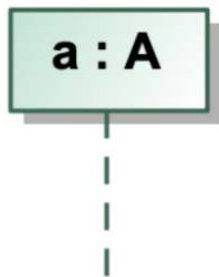


## Identificere kritisk vej

- Konstruerer Gantt diagrammer
  - Brug et værktøj
  - Paralleliser
  - Vær opmærksom på bindinger
  - Find kritisk vej
- Kritisk vej kan ændre sig
  - Opdater
  - Monitorer
- Sæt alle sejl på kritisk vej
  - Bedste ressourcer

## objekt kendskab / reference

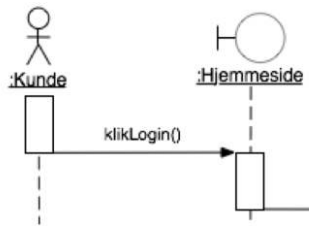
- Attributter
  - B er en Attribut i A
- Parametre
  - B er en parameter til en metode i A
- Lokale variable
  - B er en lokal variabel i en metode i A
- Global variabel
  - B er en global variabel



## Design elementer

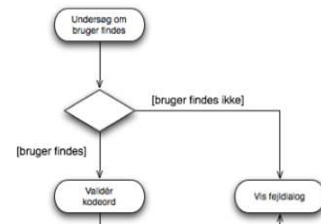
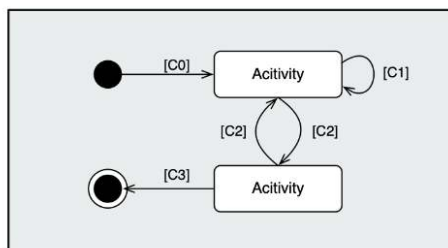
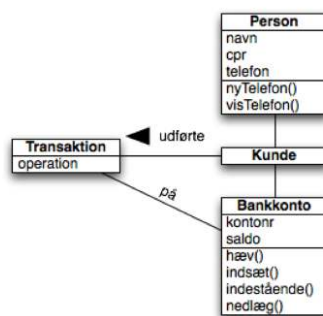
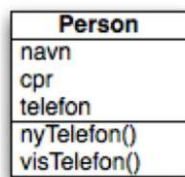
- Sekvensdiagram
  - Beskeder mellem objekter
- Klasse diagram
  - Struktur af klasser
- Aktivitets diagram
  - Algoritmer
- Tilstands diagram
  - Overgange mellem tilstande

er



r

ide



## Software kvalitet

- Imødekommer krav
  - (hvis) kravene er gode
  - Utility
- Brugbart
  - Usability
- Velstruktureret
  - Vedligeholdelsesvenligt
- Lever op til standarder
  - Programmering, proces, dokumentation

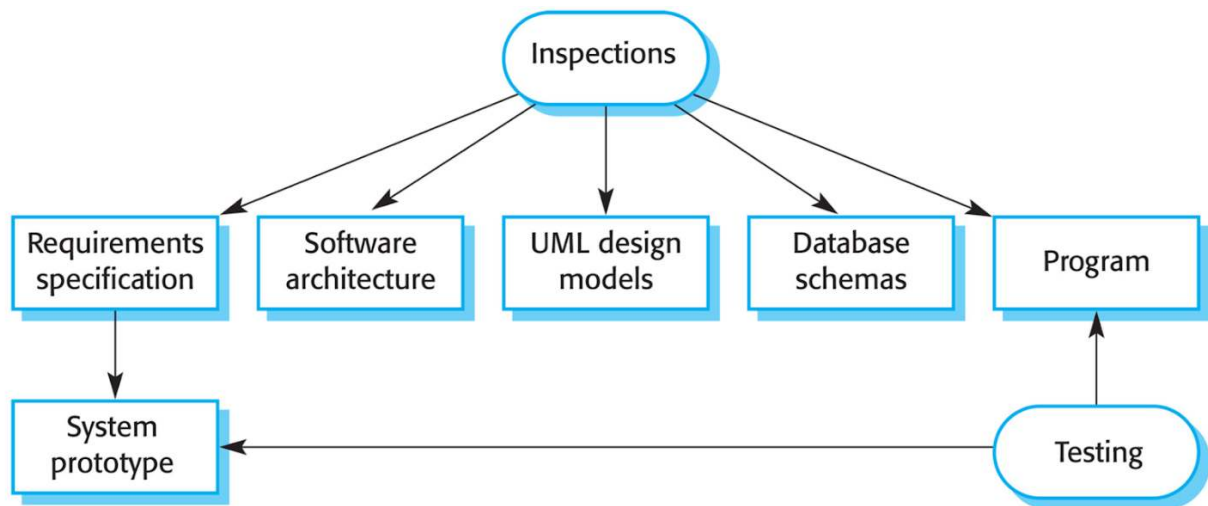
## Verificering og validering

- Test, inspektion og review
  - Del af verificering og validering
- Validering

- Kontrol af at produktet lever op til behovet
- Verificering
  - Kontrol af at produktet lever op til kravene
- Validering og verificering skal passe sammen
  - Hvis krav og behov ikke matcher sammen har man et problem

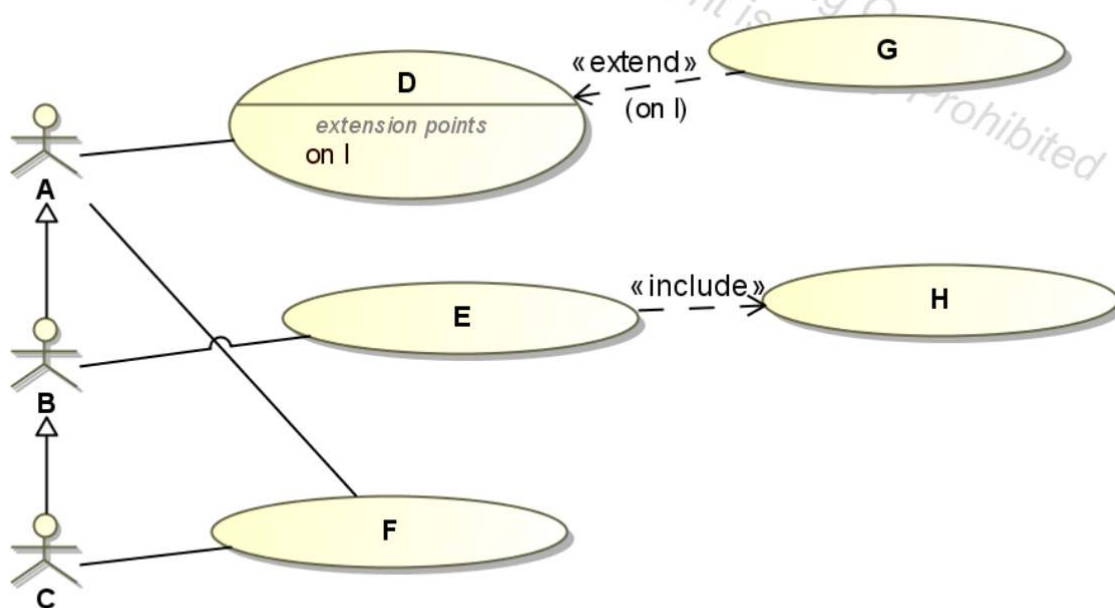
## Reviews and inspections

- Quality assurance
  - Quality of deliverables - krav, software, documentation etc



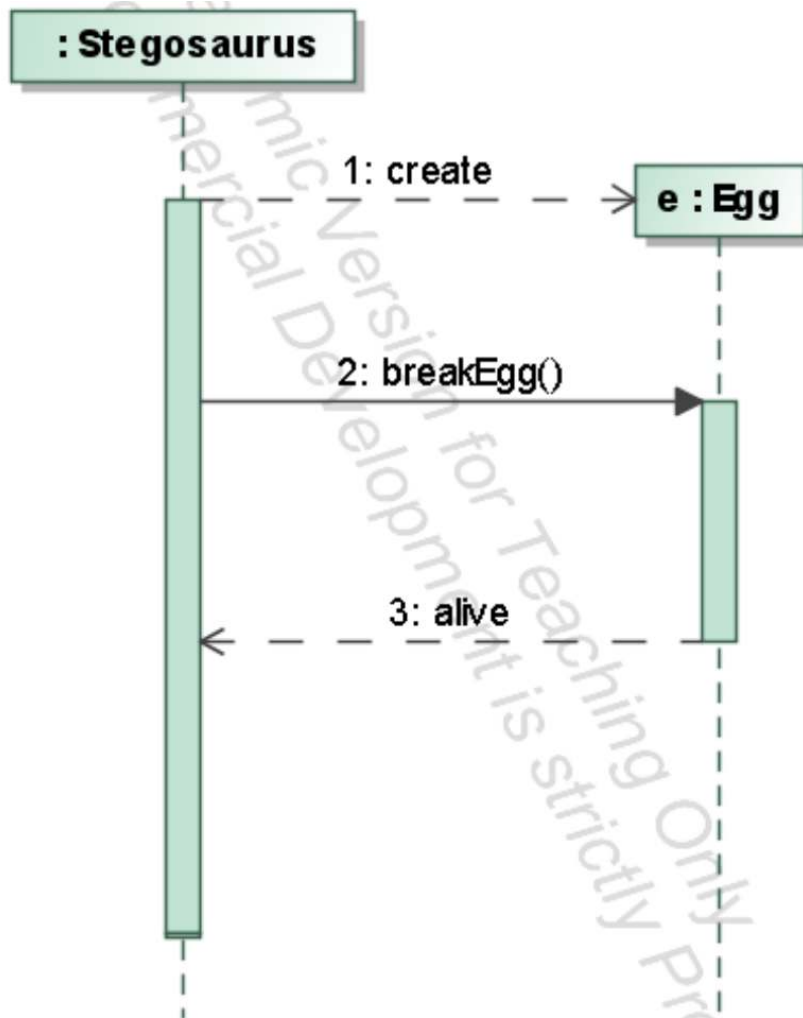
Copyright ©2016 Pearson Education, All Rights Reserved

## Svar på spørgsmål fra prøveeksamen der måske kan bruges



Choose one answer

- ☒ C kan interagere med G
- ☐ A kan interagere med alle use cases
- ☐ B kan ikke interagere med F
- ☐ E kan udføres uafhængigt af H
- ☐ G kan udføres uden D



Choose one answer

☒ `public class Stegosaurus {`  
  
    `public doStegosaurusStuff() {`  
        `Egg e = new Egg();`  
        `boolean alive = e.breakEgg();`  
  
    `}`  
`}`

Side 12 / 45

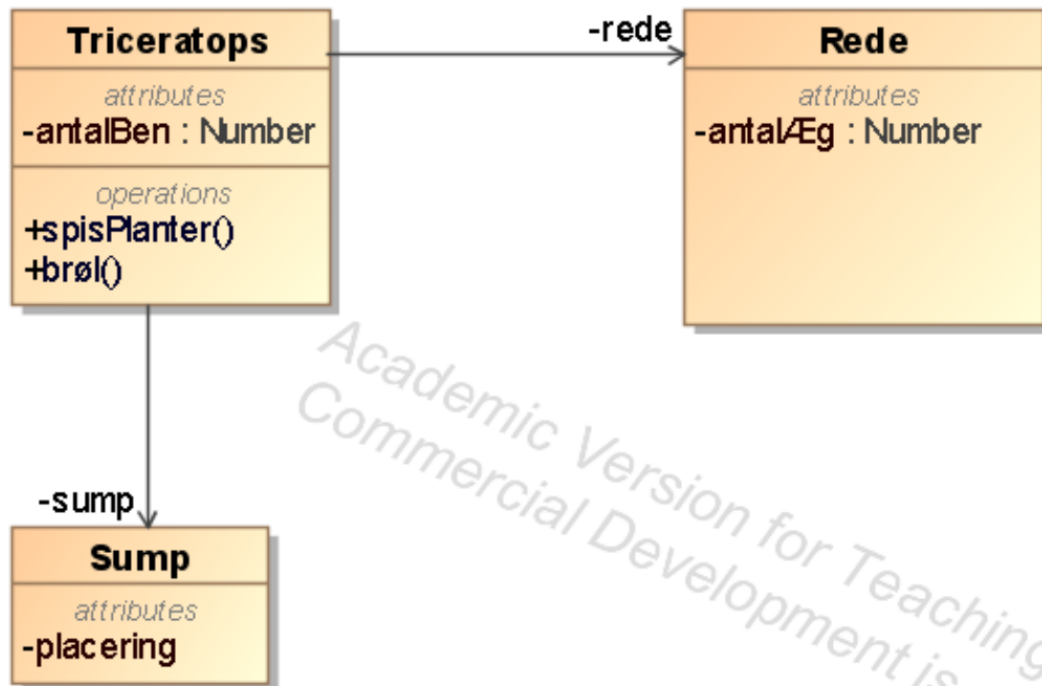


## Supplementary specification er en del af

Choose one answer

- ☐ Design
- ☒ Requirements
- ☐ Implementation
- ☐ Business Modeling
- ☐ Project Management

Hvor mange felter kræves der i klassen Triceratops for at implementere dens associationer og attributter?

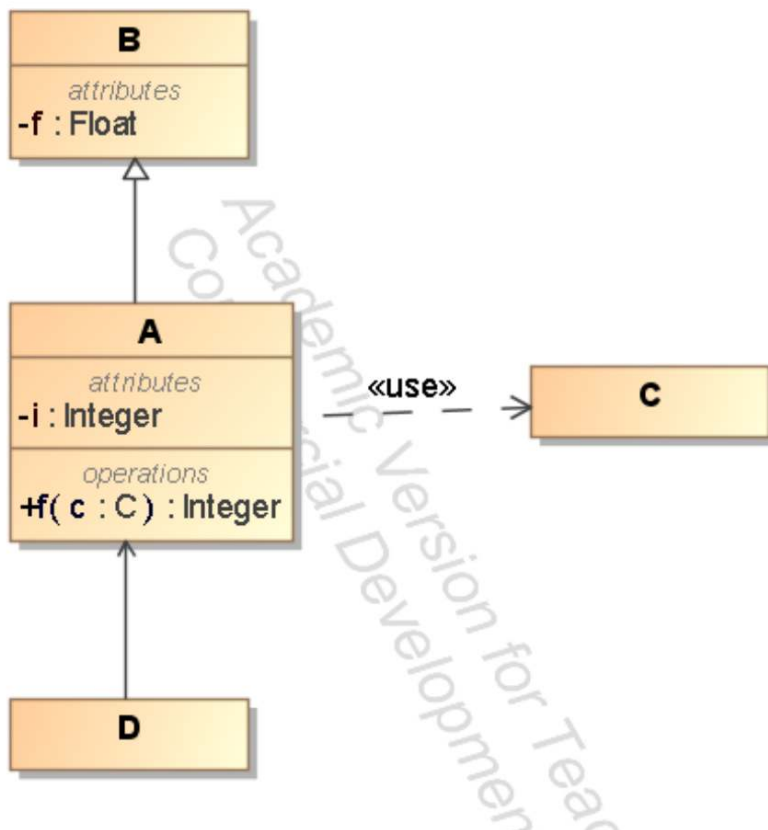


Choose one answer

☐ 0

☒ 3

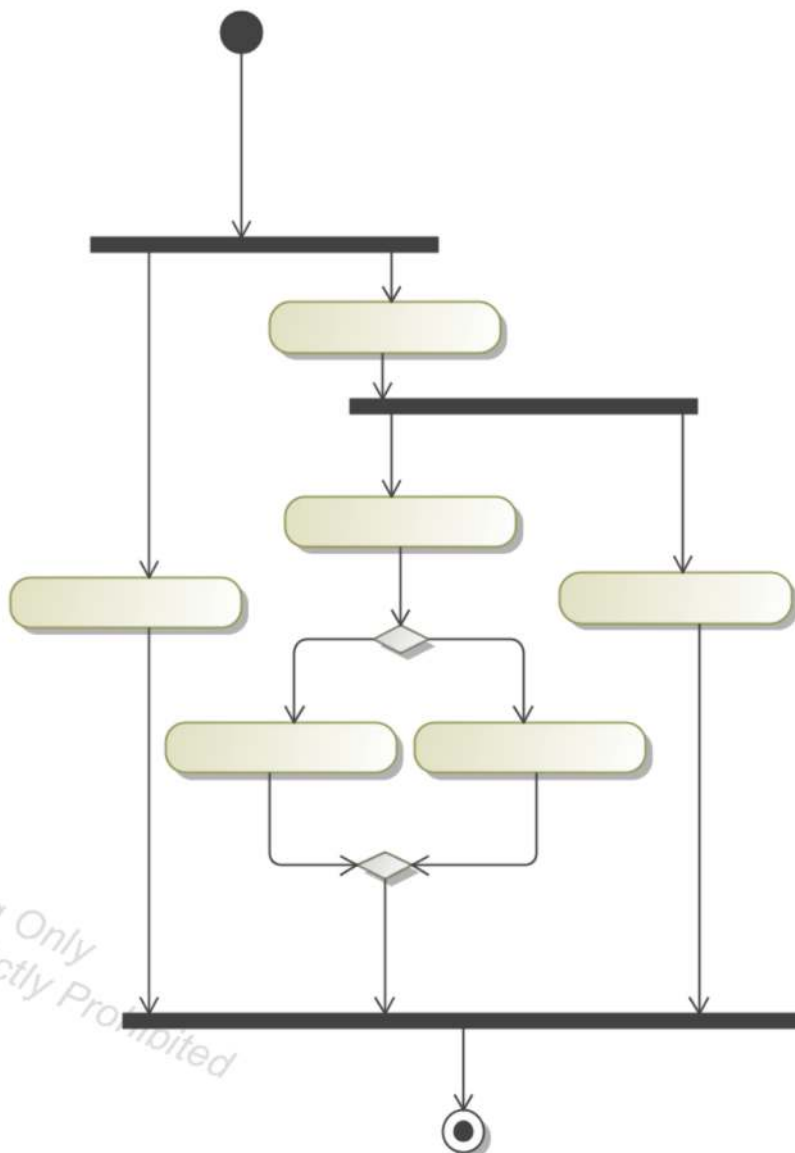
Hvilken kode svarer til følgende diagram?



✓ `public class A extends B{  
 private Integer i;  
  
 public Integer f(C c){  
 Integer i = c.g();  
 return i;  
 }  
}`



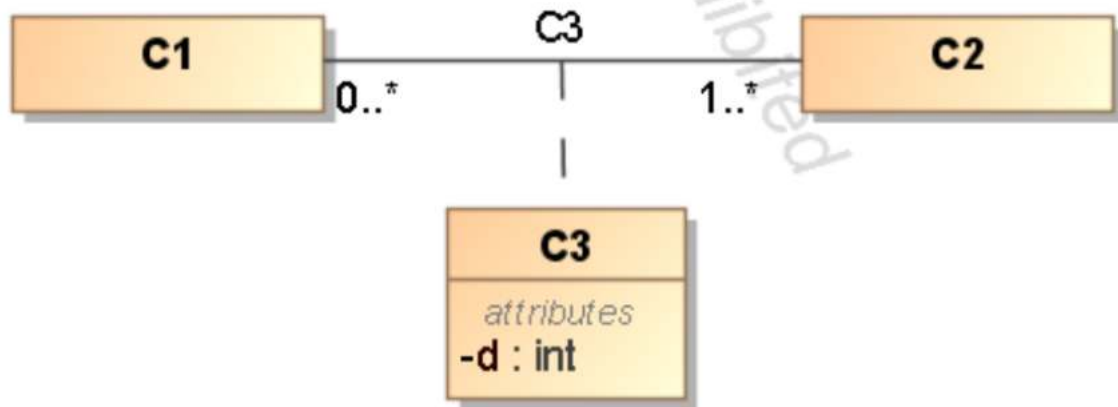
Hvad er det maksimale antal samtidige actions/handlinger i dette diagram?



Choose one answer

- ☐ 0
- ☐ 4
- ☐ 2
- ☒ 3

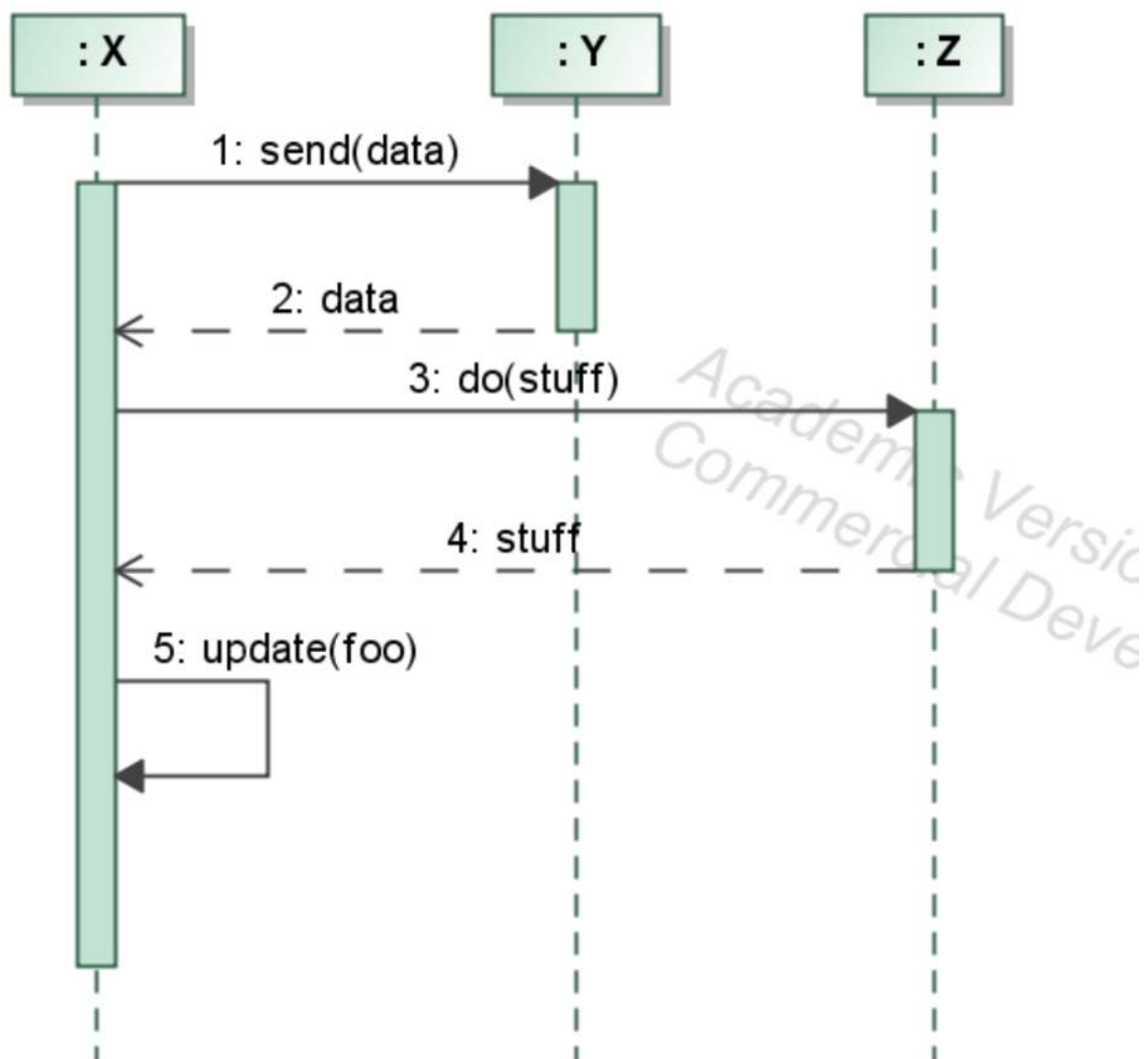
Hvad gælder her?



Choose one answer

- ☐ For hver C1 er der højst en C3
- ☒ Klassen C3 beskriver associationen mellem C1 og C2
- ☐ Klassen C3 implementerer både C1 og C2
- ☐ C3 er et interface
- ☐ C3 er en refleksiv association

Hvad er sandt?



Choose one answer

- ☐ Objektet af typen Y laver et refleksivt kald
- ☐ Objektet af typen Y kalder objektet af typen Z
- ☒ Z implementerer do(stuff)
- ☐ Objektet af typen X kalder objektet af typen Y asynkront
- ☐ X implementerer send(data)