ATM USE CASE DIAGRAM

- Les Requirements Statements for ATM System
- Lag et use case diagram som viser use casene for ATM systemet

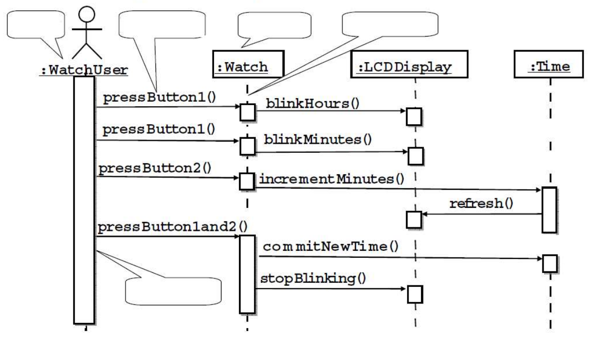
ATM USE CASE DESCRIPTION

- Gitt følgende Use Case Diagram for an ATM.
- Gjennomfør følgende Use case description for Withdraw money

Sequence diagram

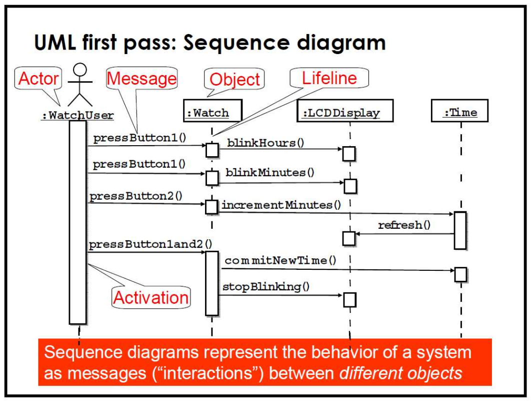
- Fyll inn de tomme boblene
- Lag klassediagram for følgende
- Hvilke tre typer kan klasser deles inn i?
- Kan du gi en beskrivelse av de tre typene?

UML first pass: Sequence diagram



Løsning

- Typer klasse:
  - Controller class
  - Boundry class
  - Entity class

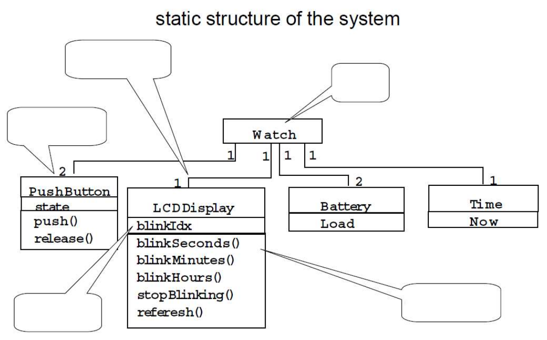


UML first pass: Sequence diagram

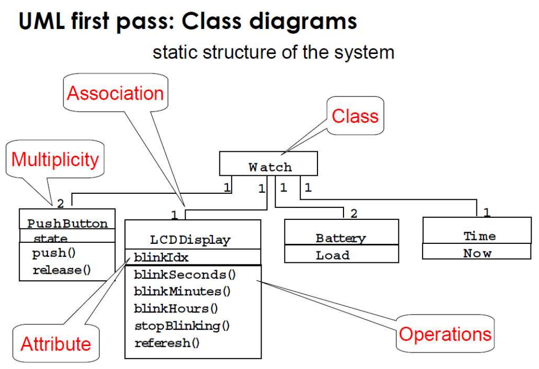
Sequence diagrams represent the behavior of a system as messages (“interactions”) between *different objects*

Class diagram

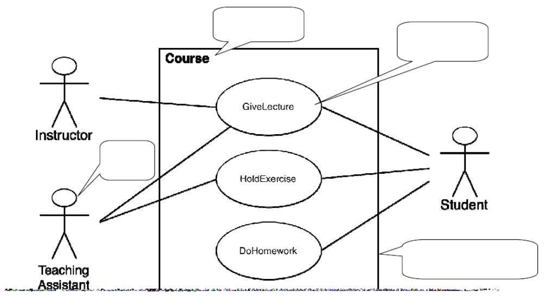
- Fyll ut de tomme boblene



Class diagram løsning

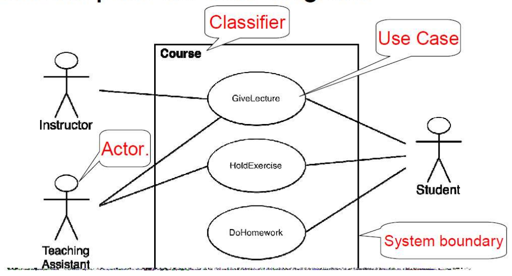


Use Case diagram løsning



Use Case diagram

UML first pass: Use case diagrams



Use case diagrams represent the functionality of the system from user's point of view

UML Diagrammer

- Hva er hensikten med et use case diagram?
- Hva er hensikten med en class diagram?
- Hva er hensikten med et sequence diagram?

- Use case diagra describes the functional behaviour of the system as seen by the user.
- Describes the static structure of the diagram, like objects/classes, attributes, and associations.
- Describe the dynamic behavior between objects of the system.

Software engineering

- Second law of software engineering states:
  - Software should be written for people first.
- Hva ligger det i dette statementet?

- Although computers run software, the hardware quickly becomes outdated, hence software must be updated
- Useful + good software lives long
- To nurture software, people must be able to understand it

Software process

- Hvilke faser inngår i waterfall modellen?
- Hva er en av hovedulempene med waterfall processen?

- There are separate identified phases in the waterfall model:
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance
- Main drawback:
  - The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase.

Waterfall

- Kan du nevne i hvilke situasjoner waterfall kunne vært et godt alternativ?

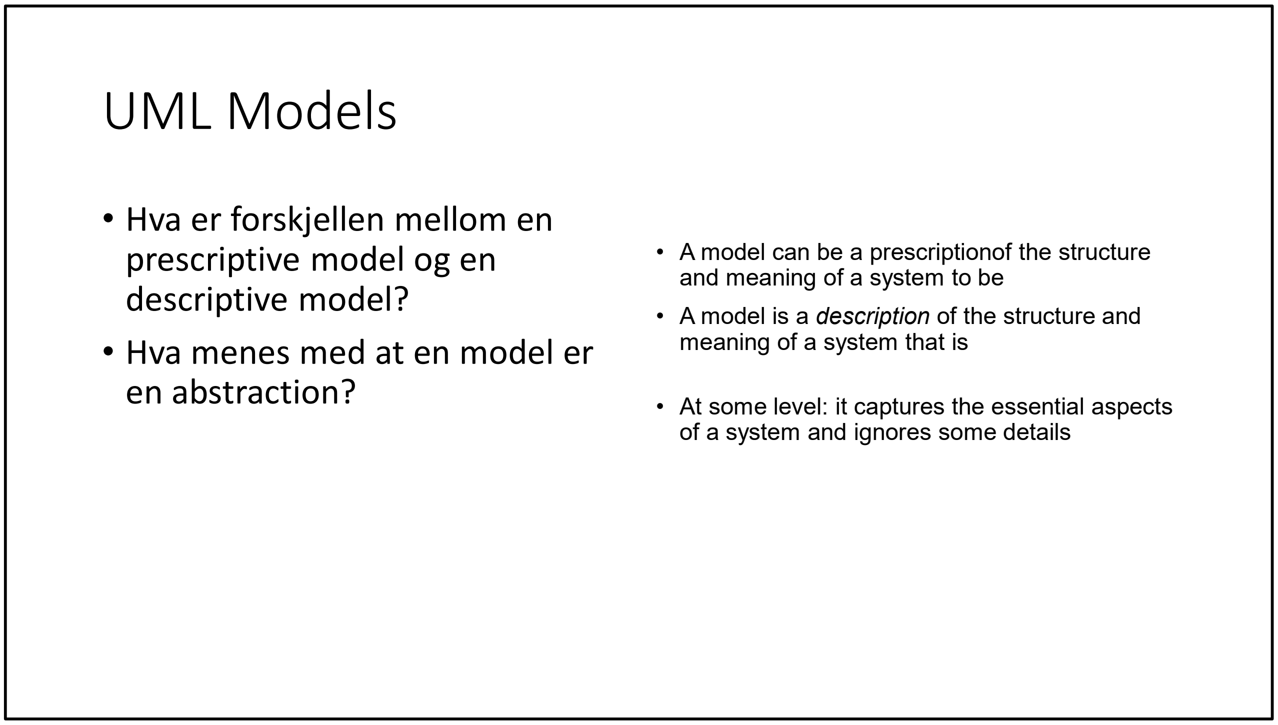
- Forslag:
- Appropriate when the requirements are well-understood and changes will be fairly limited during the design process
- In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.
- Embedded system where software interface with hardware
- Critical systems where safety and security is important
- Large software systems, many companies/partners and several sites

Agile

- Hva er noen av fordelen, men også utfordringene med agile development process?

- **conflicts with the procurement model** of many organizations, where the complete system specification is part of the system development contract.

- Difficult to predict cost
- Customeris accepting more risk with more involvement
  - Ref IBM vs Statens Vegvesen



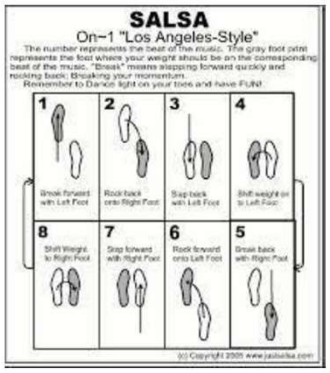
UML Models

- Hva er forskjellen mellom en prescriptive model og en descriptive model?
- Hva menes med at en model er en abstraction?

- A model can be a prescriptionof the structure and meaning of a system to be
- A model is a *description* of the structure and meaning of a system that is
- At some level: it captures the essential aspects of a system and ignores some details

Models

- I UML sammenheng. Hva vil du si om disse modellene?
- Statis vs runtime


SALSA

On−1 "Los Angeles-Style"

The number represents the beat of the music. The gray foot print represents the foot where your weight should be on the corresponding beat of the music. "Break" means stepping forward quickly and rocking back; Breaking your momentum.

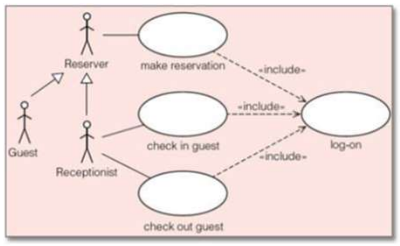
Remember to Dance light on your toes and have FUN!

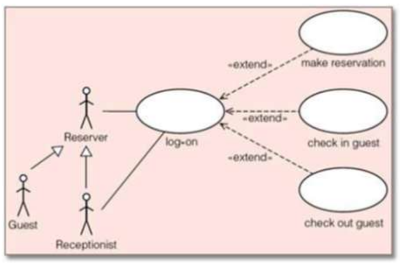
1
Break forward with Left Foot
2
Rock back onto Right Foot
3
Step back with Left Foot
4
Shift weight on to Left Foot
8
Shift Weight to Right Foot
7
Step forward with Right Foot
6
Rock forward onto Left Foot
5
Break back with Right Foot
(c) Copyright 2005 www.jasalsa.com

Use case

- Hva kan du fortelle meg om disse to Use Case diagrammene?
- Hvor mange use caser inneholder de?
- Hva er forskjellen/likhetene?

- To like diagrammer, men beskrevet på litt forskjellige måte.
- Hver inneholder 3 use caser.
- Liten forskjell, smak og behag i forhold til hvordan de er visualisert.



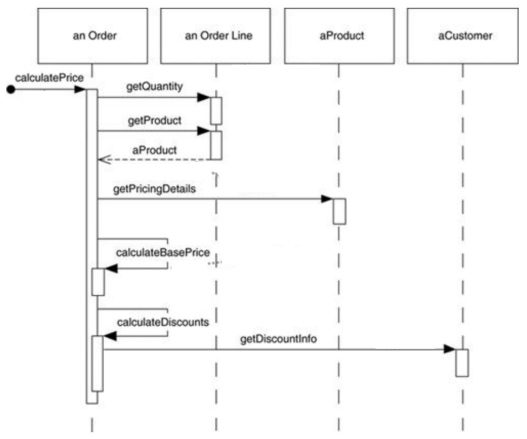


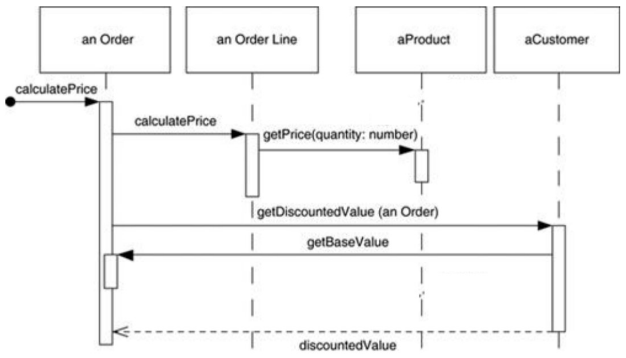
OO modelling

- I en OO verden, hva mener vi med abstraction?
- Hva mener vi med encapsulation?
- Hva er relasjoner innen OO modelling, og har du eks på slike relasjoner?

- Abstraction:
  - hide minor details so to focus on major details
  - Often said in the class: "take a step back, lock at the overall goal or achievement, and focus on that".
    - Key pad lock vs general authentication
- Encapsulation:
  - Modularity: principle of separation of functional concerns
  - Information-hiding: principle of separation of design decisions
- Relations:
  - Association: relationship between objects and classes
  - Inheritance: relationship between classes, useful to represent generalization or specialization.

Hva er forskjellen mellom disse to diagrammene?





Dette er tatt fra pensum: forskjellen ligger i at vi på venstre side har sentralisert kontroll, hvor order, (one participant) gjør all processering, og de andre objectene supply data. Mens til venstre har vi en mer de-sentralisert (distribuert) kontroll hvor processen er delt på flere deltagere – og hver deltager gjør litt hver av processen.
Det er fordeler og ulemper med begge, den til venstre er letter å følge, all processen skjer på ett sted og man slipper å jage programflowen rundt i objektene.
Fordelen med den til høyre er localize effect of change – en viktig del av OO design er å putte data and behaviour som bruker data sammen, som gir high cohesion.

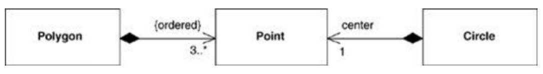
Hva tenker du når du ser?

- 1. Shep is a Border Collie.
- 2. A Border Collie is a Dog.
- 3. Dogs are Animals.
- 4. A Border Collie is a Breed.
- 5. Dog is a Species.

- Generalization.
- Her kan man snakke om:
  - sub og super classes
  - Generalization vs spesialization
  - Hva skal man se etter når man leser tekst for å finne ut om noe er i retning av generalization?
    - Phrase «is-a»
  - Hva kan være utfordringer med generalization?
    - Favor composition over inheritance.
- Hvilken forskjell er det på inheritance vs realization?
  - Funksjonalitet vs kontrakt

UML Diagram

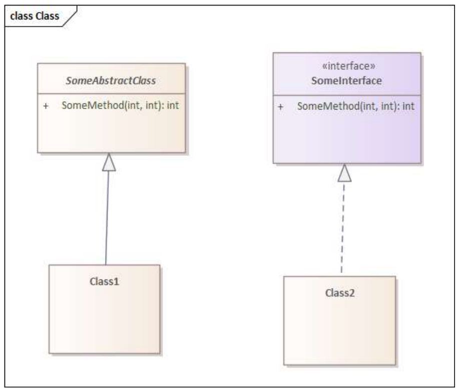
- Hva er det dette diagrammet forteller oss?



- Hovedtanken er at Polygon og Circle ikke deler Point – hver av den har eksklusive ownership.
- At Point skal slettes når Polygonet eller Circle'en slettes.
- At Point ikke kan eksistere «alene».

Interface and abstract classes

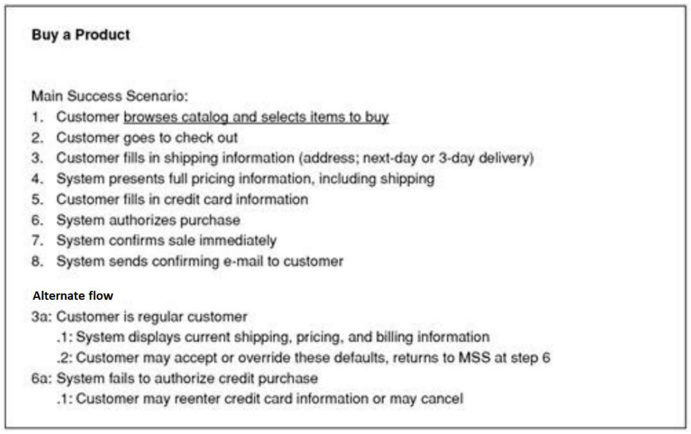
- Hva er forskjellen mellom disse to?



- SomeAbstractClass forventes å ha en implementasjon av SomeMethod.
- SomeAbstractClass kan ikke instansieres.
- SomeInterface har ingen implementasjon av SomeMethod, det er bare en kontrakt.
- Class2 må ha en beskrivelse av alle metodene i klassen.
- Class1 trenger bare å ha en beskrivelse av metoden hvis den skal override metoden.

Fra Use Case til Interaction diagram

- Gitt denne Use Case description.
- Lag et use case diagram.
- Lag et interaction diagram.
- Hva er en pre-condition i et use case?
- Har du et eks på hva som kunne vært en pre-cond. Her?
- Hva er en business trigger? Har du et eks på hva som kunne være en business trigger her?



Buy a Product

Main Success Scenario:
1. Customer browses catalog and selects items to buy
2. Customer goes to check out
3. Customer fills in shipping information (address; next-day or 3-day delivery)
4. System presents full pricing information, including shipping
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming e-mail to customer

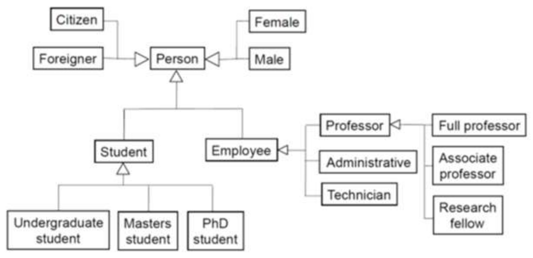
Alternate flow

3a: Customer is regular customer
 .1: System displays current shipping, pricing, and billing information
 .2: Customer may accept or override these defaults, returns to MSS at step 6

6a: System fails to authorize credit purchase
 .1: Customer may reenter credit card information or may cancel

Studentne er vant til å se dette som to kolonner, hvor actor er venstre kolonne og systemet er høyre kolonne. Dette er litt forenklet her, og det må man si til studenten.
Den skal være en oval, og den skal hete Buy a Product. Use caset skal være inni en boundry, og det skal være en user på utsiden.

Modell

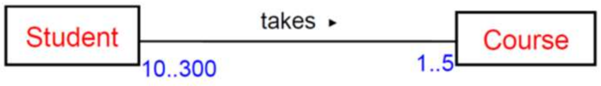


- Hva vil du si om følgende modell? (ikke om selve innholdet, men hva den skal representere).
- Hvilken metode ville du brukt for å komme fram til den?
- Hvordan ville du gått fram for å finne riktige attributter?
- Hvordan ville du gått fram for å finne riktige funksjoner?

- Conceptuell model fra domenet
- OOA
- In conjunction with interaction diagram

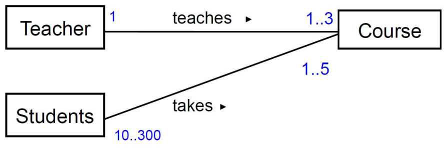
Class og association

- Hva ønsker jeg å fortelle utvikleren med denne modellen?



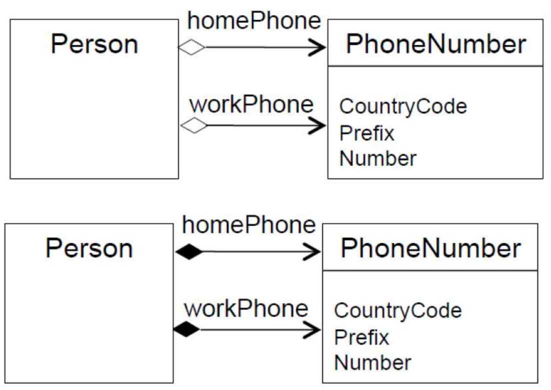
- Utvid modellen slik at en lærer underviser 1 til 3 kurs, og at et kurs undervises av en og bare en lærer.

- A Student can take up to five Courses
- Every Student has to be enrolled in at least one course
- Up to 300 students can enroll in a course
- A class must have at least 10 students



Model

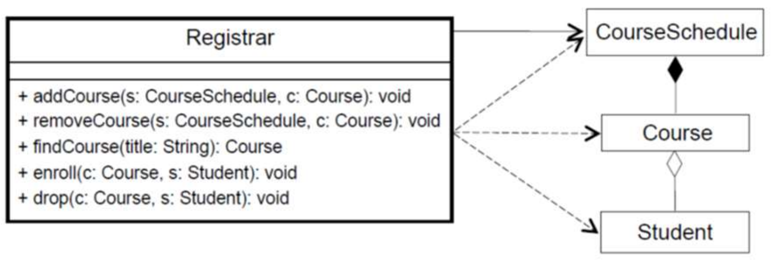
- Hva er forskjellen på disse to?



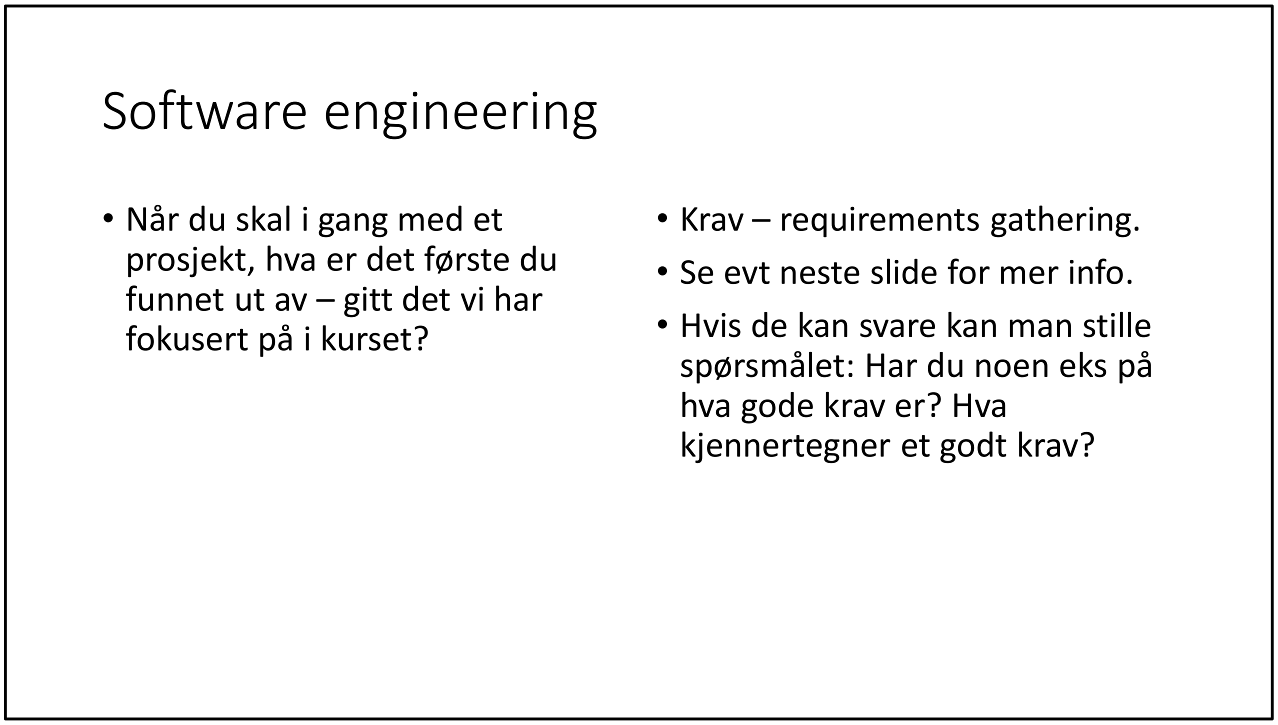
- Aggregation:
  A Person has two phone numbers. The numbers are independent from the Person. The PhoneNumber can exist as an object outside the scope of a Person.
- Warning–make code oft his UML and the code will be the same. The model holds a message to the development team.
- Composition:
  A person has two phone numbers. The numbers are dependent of Person. The PhoneNumbers cannot exists as objects outside the scope of a Person.

Model

- Kan du forklare diagrammet?


Registrar
+ addCourse(s: CourseSchedule, c: Course): void
+ removeCourse(s: CourseSchedule, c: Course): void
+ findCourse(title: String): Course
+ enroll(c: Course, s: Student): void
+ drop(c: Course, s: Student): void
CourseSchedule
Course
Student

Her ligger noe av forskjellen i å forstå forskjellen mellom assosiasjonene. Composition vs Aggregation. Strong dependency vs weak dependency.
I koden ville f.eks Registrar ha en private varibel CourseSchedule, men ikke Course og Student. Den er kun avhengig av Course og Student pga funksjonene.



Software engineering

- Når du skal i gang med et prosjekt, hva er det første du funnet ut av – gitt det vi har fokusert på i kurset?

- Krav – requirements gathering.
- Se evt neste slide for mer info.
- Hvis de kan svare kan man stille spørsmålet: Har du noen eks på hva gode krav er? Hva kjennertegner et godt krav?

Eks på gode krav.
•Complete
•Containall theinformationfor thereaderto understand it
•UseTBD (to be decided/determined) to indicategap or lackofinformation
•Correct
•Must accuratelydescribethecapabilitythatwillmeetthestakeholders need
•Go to sourceofrequirementsto verifycorrectness
•Feasible

•Must be possibleto implementeachrequirementwithintheknowncapabilites and limitatinofthesystem.
•Make a prof-of-conceptifnecessary
•Necessary
•Must be supportedby a business value
•Prioritized
•Decidewhicharemore importantthanothers.
•Unambigous
•Natural languageis proneto ambiguity, ifthereis more thanonewayto interpret a requirement, it willbe interpretedthewrongwayat somepoint
•Verifiable
•Cana testeerdevise tests or otherform ofverificationapproachesto determinewhethertherequirementis properlyimplemented?
•Tracable
•CanI trace therequirementback to itsorigin, and forward to itsimplementation; includingtest descriptionand test execution?

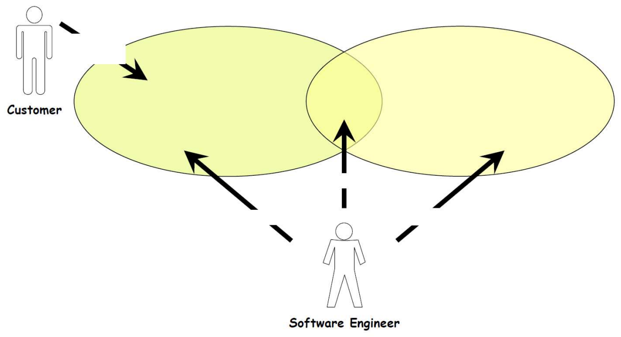
Software engineering

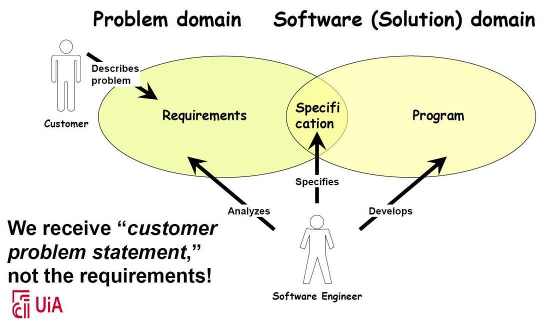
- Har du eks på hvordan man kan gjøre krav innsamlig? (requirements elicitation)

- Interviews
  - Traditional soure of requirements
  - Interviews with an expert can excel your application domain
  - One to on or small groups usually provides more buy in
- Workshop
  - Encourage stakeholder to collaborate in finding requirements
  - Activates several types of stakeholders.
  - More resource intensive than intervewis
- Focus Groups
  - Group of representatives that will aid in the requirement elicitation
  - Best with a highly diverse group
- Observation
  - nterviews may lead to under precise descriptions people often tell half the tail) hard to remember
  - complex details
  - Observertions can validate information collected from other sources
  - Time consuming , may not be suitable for every task
  - Can be both silent and interactive . « Can you explain why you did that ?»
- Questionnaires
  - Survey large groups
  - Inexpensive , but challenging to define well written questions
  - Often a good input to other elicitation techniques

Software engineering

- Kan du forklare følgende figur?

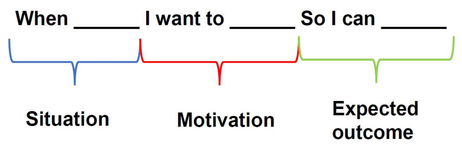


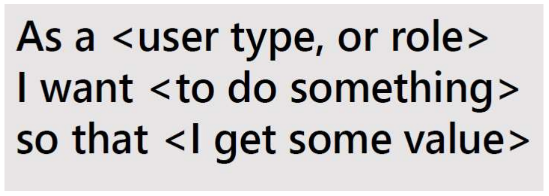


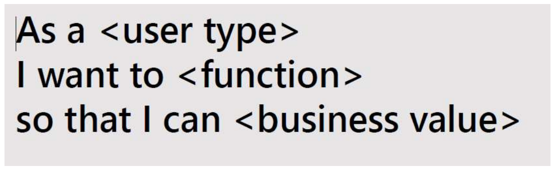
We receive "*customer problem statement*," not the requirements!

User stories

- Hvis du skulle forklare for andre hvordan de skulle bygge opp en user story – hva ville du sagt?

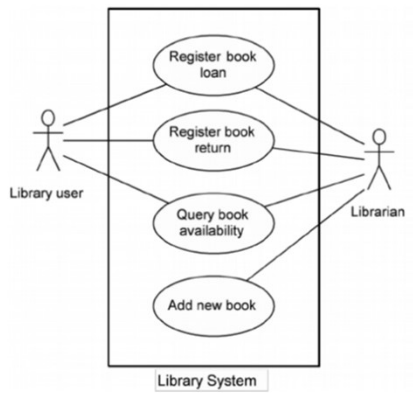



As a <user type, or role>
I want <to do something>
so that <I get some value>


As a <user type>
I want to <function>
so that I can <business value>

Use case

- Hvem er primary actor i dette tilfelle?



- Who is the primary actor:
  - Can the Library User initiate this with the system itself or dose the
  - Library User ask the Librarian to initate it?
- Who is the Supporting actor
  - If the Librarian must initate the register book, then Library user is the supporting actor
  - If the Library User can initate the use case, but sometimes need help from the Librarian , then the Librarian is the supporting actor

Perimary actor: The primary actor of a use case is the stakeholder that calls upon the system to deliver one of its services. The primary actor has a goal with respect to the system, one that can be satisfied by its
operation. The primary actor is often, but not always, the actor who triggers the use case.

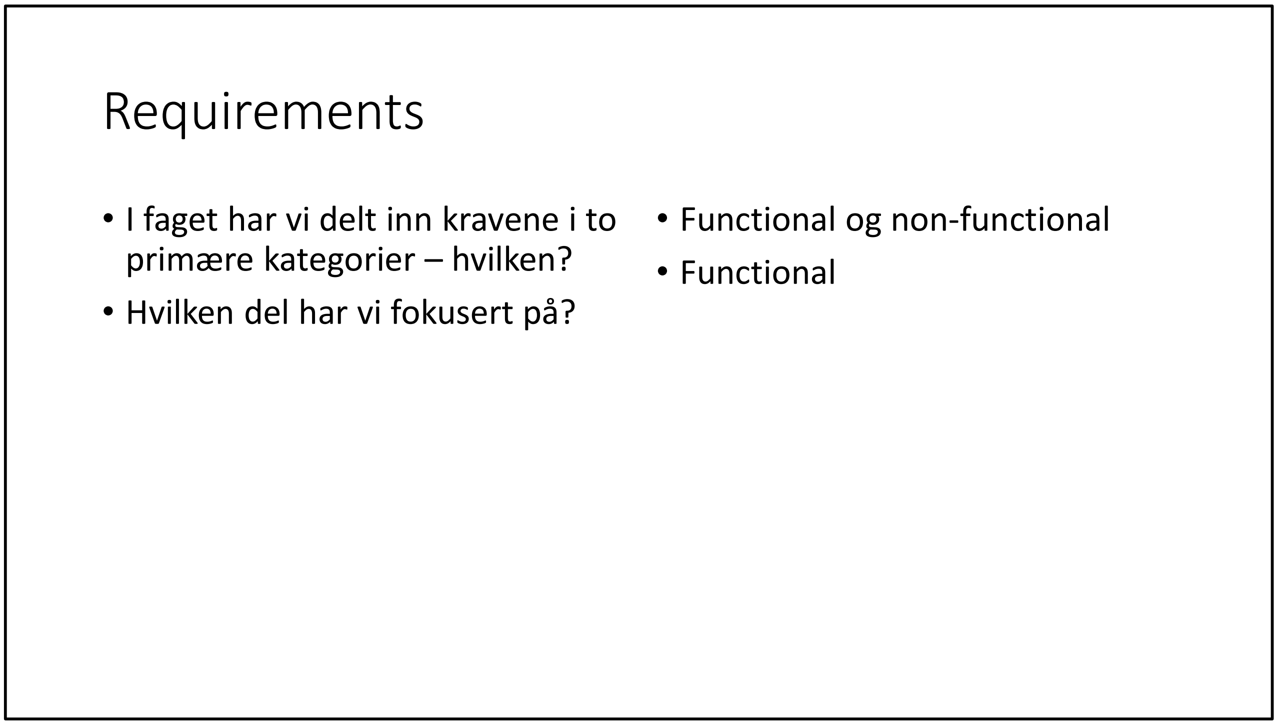
Supporting actor: A supporting actor of a use case is an external actor that provides a service to the system under design. It might be a high-speed printer, it might be a web service, or it might be a group of
humans who have to do some research and get back to us

(for example, the coroner's office, which
provides the insurance company with confirmation of a
person's death). We used to call this a
secondary actor, but people found the term confusing.
More people are now using the more natural
term, supporting actor.

Software engineering

- Hva er en assumption?

- An assumption:
  - is a statement that is believed to be true, or accepted as true in the absence of proof or definitive knowledge a fact or statement that is taken for granted
- «It is assumed that the ATM should be placed inside a shopping mall , hence the project will not consider wheather proffing it.»
- As long as an assumption is not verified , it is considered a risk.



Requirements

- I faget har vi delt inn kravene i to primære kategorier – hvilken?
- Hvilken del har vi fokusert på?

- Functional og non-functional
- Functional

Traceability

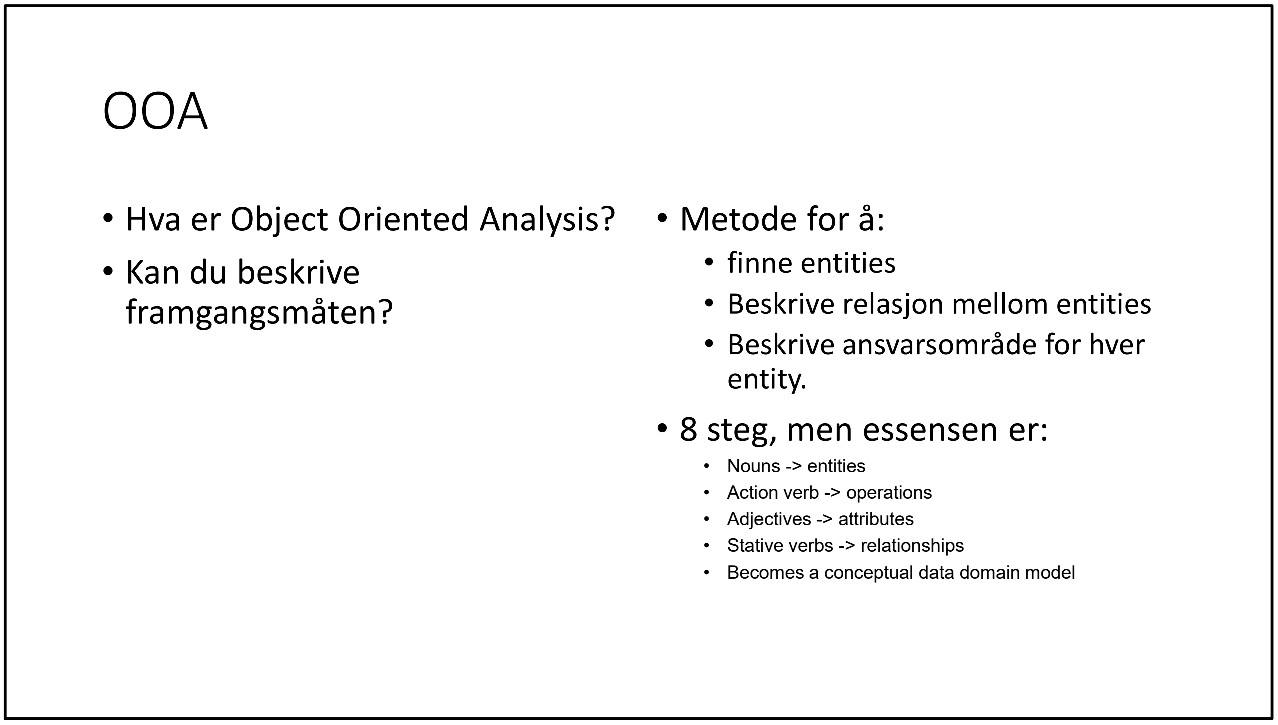
- Hva er traceability og hvorfor er det viktig?

- Vite at man har fått med seg alle kravene i modellen og koden.
- Vite at all funksjonalitet stammer fra et user krav – no gold plating
- Kunne tracke endringer i systemet.
- Kunne tracke bug og test cases i systemet.

Modelling

- Hva er hensikten med modelling?
- Kjenner du til en modelleringsteknikk?
- Hva er forskjellen på analyse og design?

- Analysis and modeling identifies the system elements (inside the system) needed to solve the problem (i.e., meet the requirements)
- The goal of domain modeling is to understand how system-to-be will work (or how the real worldwork)
  - Requirementsanalysis determined how users will interact with system-to-be (external behavior)
  - Domain modeling determines how elements of system-to-be interact (internal behavior) to produce the external behavior
- Modelleringsteknikk brukt I kurset er OOA.
- Analyse definerer problem domain, design definerer solution domain eller hvordan problem domene mapper til splution domenet.



OOA

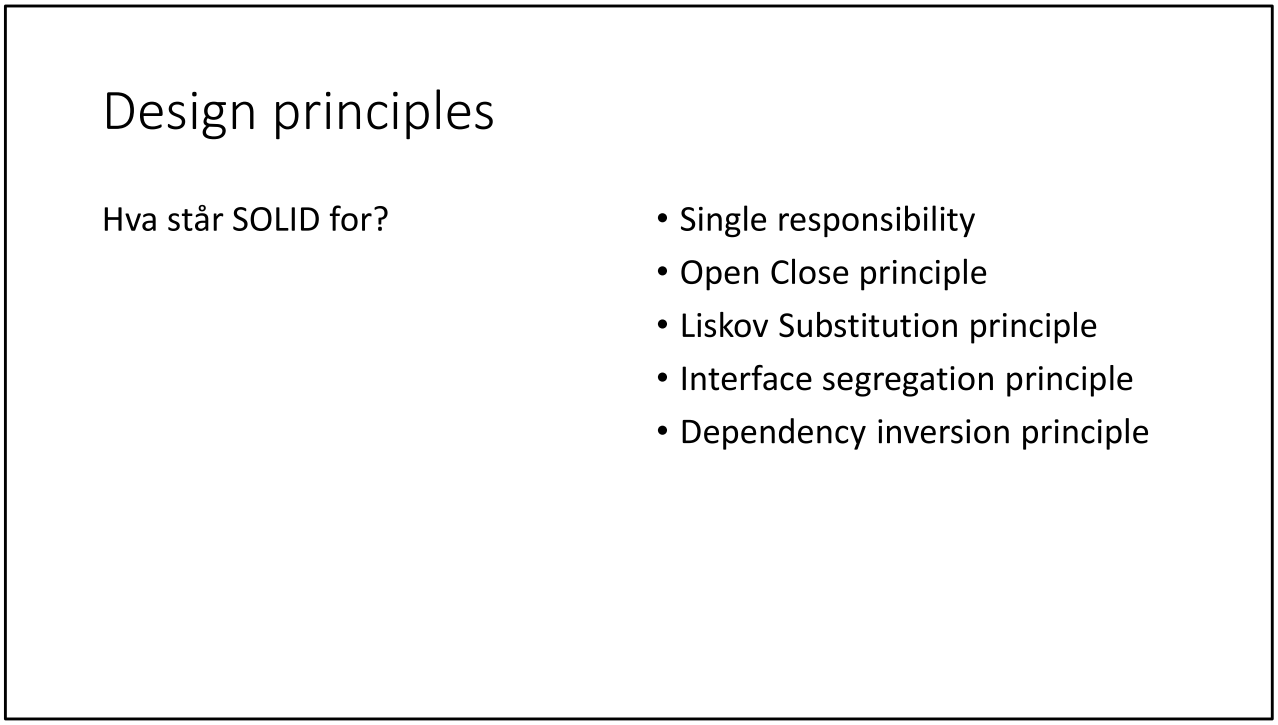
- Hva er Object Oriented Analysis?
- Kan du beskrive framgangsmåten?

- Metode for å:
  - finne entities
  - Beskrive relasjon mellom entities
  - Beskrive ansvarsområde for hver entity.
- 8 steg, men essensen er:
  - Nouns -> entities
  - Action verb -> operations
  - Adjectives -> attributes
  - Stative verbs -> relationships
  - Becomes a conceptual data domain model

Controller class

- Hva er en kontroller class?
- Hvor finner du informasjonen om en kontroller klass?

- This is the class that is responsible for running or driving the use case. In the end you will probably combine the controllerclass from several use cases, but initially create one for each use case.
- Who should be responsible for handling an input system event?
- •There is (usually) only one controller class for each use case.
- This controller class is probably not described in the use case description.



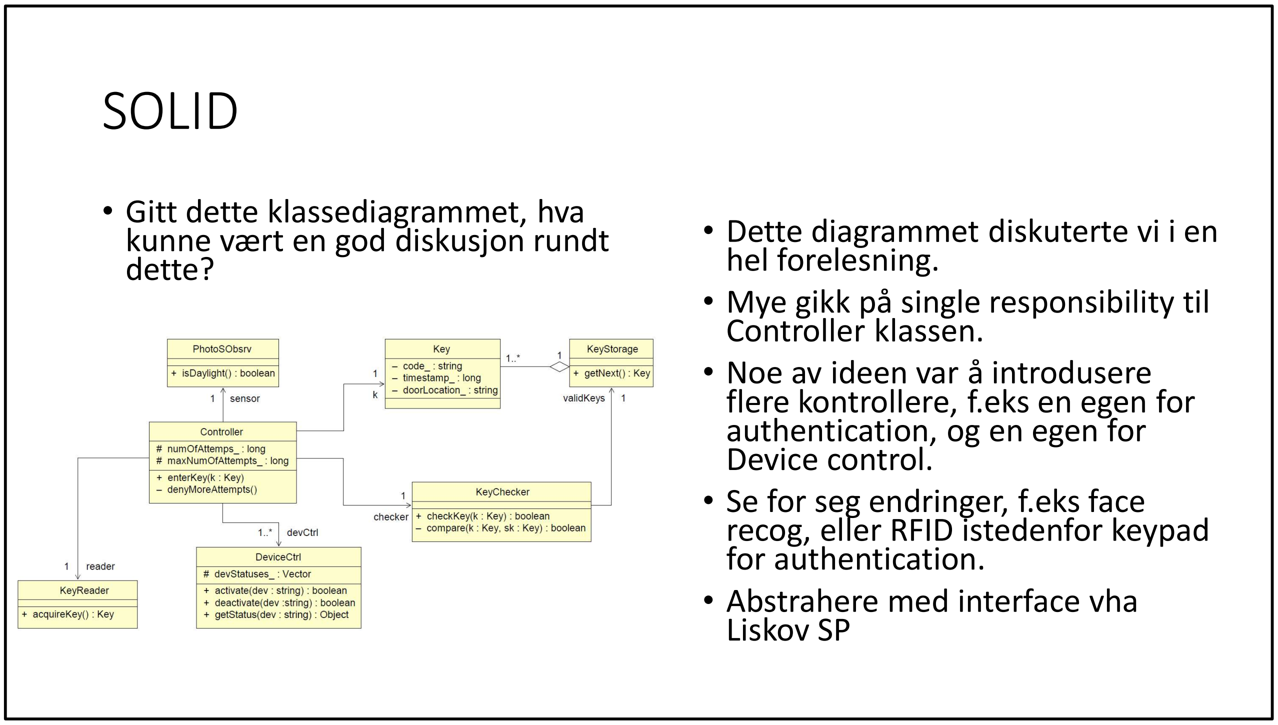
Design principles

Hva står SOLID for?

- Single responsibility
- Open Close principle
- Liskov Substitution principle
- Interface segregation principle
- Dependency inversion principle

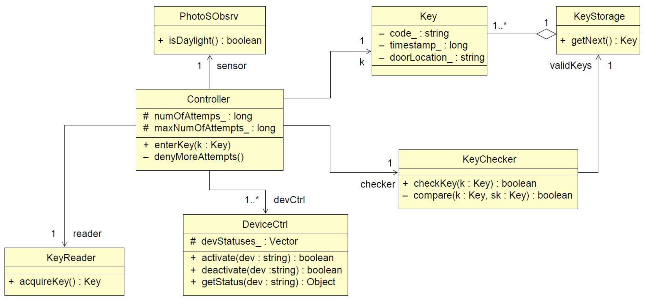
•**S**ingle responsibility principle (SRP)
•Every class/module should have only one reason to be changed –the class/module should have only one responsibility.
•If class/module "A" has two responsibilities, create new classes/module "B" and "C" to handle each responsibility in isolation, and then compose "A" out of "B" and "C"
•**O**pen/closed principle (OCP)
•Every class should be *open for extension* (derivative classes), but *closed for modification*

(fixed interfaces)
•Put the system parts that are likely to change into implementations (i.e.*concrete classes*) and define *interfaces*around the parts that are unlikely to change (e.g.*abstract base classes*)
•**L**iskovsubstitution principle (LSP)
•Every implementation of an interface needs to fully comply with the requirements of this interface (requirements determined by its clients!)
•Any algorithm that works on the interface, should continue to work for any substitute implementation
•Every subclass (or module) or derived class should be substitutable for their base or parent class.
•**I**nterface segregation principle (ISP)
•A client should never be forced to implement an interface that it does not use, or client should not be forced to depend on methods they do not use →segregate you interfaces.
•Keep interfaces as small as possible, to avoid unnecessary dependencies →only keep coherent methods in an interface.
•Ideally, it should be possible to understand any part of the code in isolation, without needing to look up the rest of the system code
•**D**ependency inversion principle (DIP)
•Instead of having concrete implementations communicate directly (and depend on each other), *decouple*



SOLID

- Gitt dette klassediagrammet, hva kunne vært en god diskusjon rundt dette?



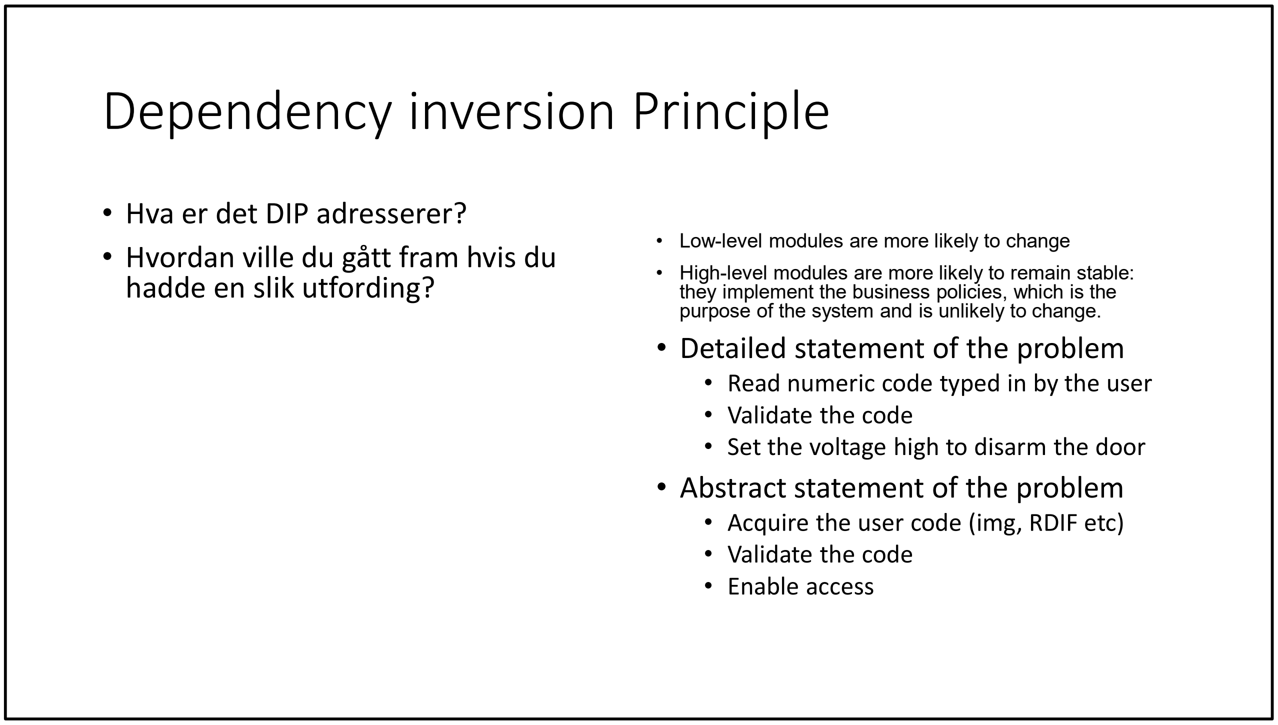
- Dette diagrammet diskuterte vi i en hel forelesning.
- Mye gikk på single responsibility til Controller klassen.
- Noe av ideen var å introdusere flere kontrollere, f.eks en egen for authentication, og en egen for Device control.
- Se for seg endringer, f.eks face recog, eller RFID istedenfor keypad for authentication.
- Abstrahere med interface vha Liskov SP

Any nontrivial design is a **compromise** between the desired and the possible
•Design principles may contradict each other:
•Shortening the communication chain (*expert doer*) tends to concentrate responsibilities to fewer objects (*low cohesion*)
•Minimizing the number of responsibilities per object (*high cohesion*) tends to increase the number of dependencies (*strong coupling*)

Design principles vs Patterns

- Hva er forskjellen på design principles og patterns?

- Design principles diagnose the problem (finds them)
- Patterns addresses the problems (solves them).



Dependency inversion Principle

- Hva er det DIP adresserer?
- Hvordan ville du gått fram hvis du hadde en slik utfording?

- Low-level modules are more likely to change
- High-level modules are more likely to remain stable: they implement the business policies, which is the purpose of the system and is unlikely to change.
- Detailed statement of the problem
  - Read numeric code typed in by the user
  - Validate the code
  - Set the voltage high to disarm the door
- Abstract statement of the problem
  - Acquire the user code (img, RDIF etc)
  - Validate the code
  - Enable access

Instead of high-level module (policy) depending on low-level module (mechanism/service/utility):
•High-level module defines its desired interface for the low-level service (i.e., high-level depends on itself-defined interface)
•Lower-level module depends on (implements) the interface defined by the high-level module
•➔*Dependency inversion*(from low to high, instead the opposite)

11

Single Responsibility Principle

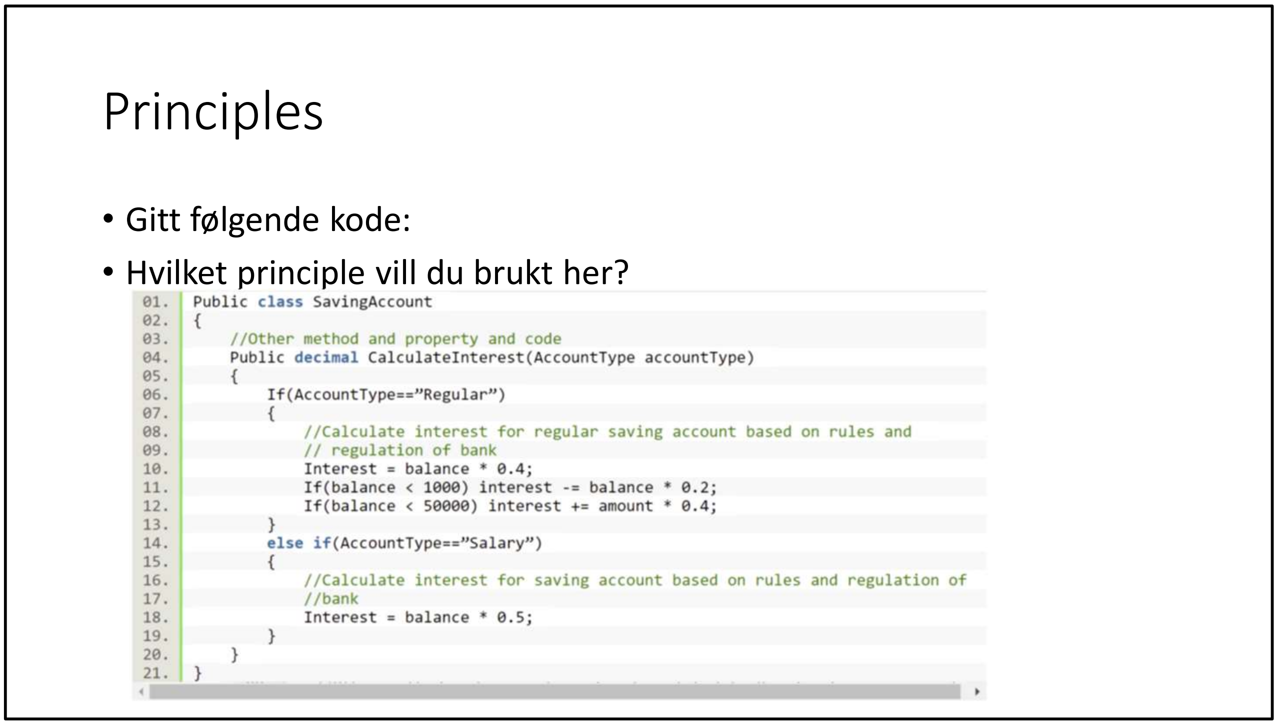
- Kan du nevne noen fordeler med SRP?

- Reduction in complexity of a code
  - A code is based on its functionality. A method holds logic for a single functionality or task.
  - So, it reduces the code complexity.
- Increased readability, extensibility, and maintenance
  - As each method has a single functionality so it is easy to read and maintain.
- Reusability and reduced error
  - As code separates based functionality so if the same functionality uses somewhere else in an application then don't write it again.
- Better testability
  - In the maintenance, when a functionality changes then we don't need to test the entire model.
- Reduced coupling
  - It reduced the dependency code. A method's code doesn't depend on other methods.

Open/Close Principle

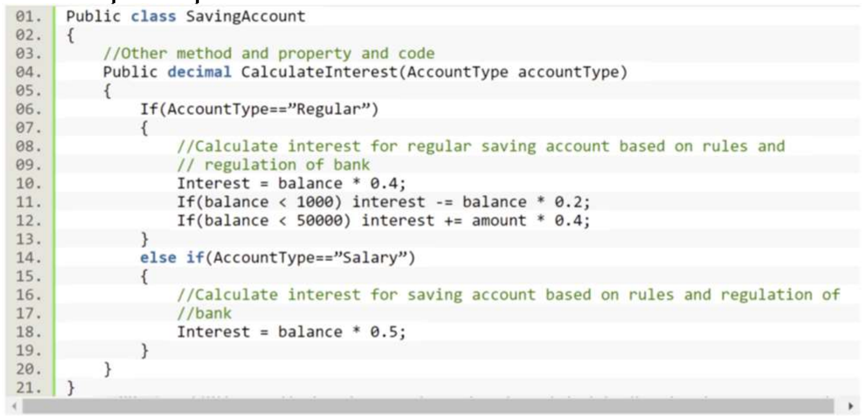
- Kan du forklare Open/Close principle?

- Every class should be *open for extension* (derivative classes), but *closed for modification* (fixed interfaces)
- Put the system parts that are likely to change into implementations (i.e.*concrete classes*) and define *interfaces*around the parts that are unlikely to change (e.g.*abstract base classes*)
- A wall outlet adapter:
  - An adapter in the wall is always closed for modification, in other wordswe cannot change it once it is fitted or extended if we want more.
  - But an adapter always provides a method of extension, so we can plug in an extension board of an adapter for more adaptation.
  - Soyou plug in an extension board and extend an existing electric adapter fitted in wall.



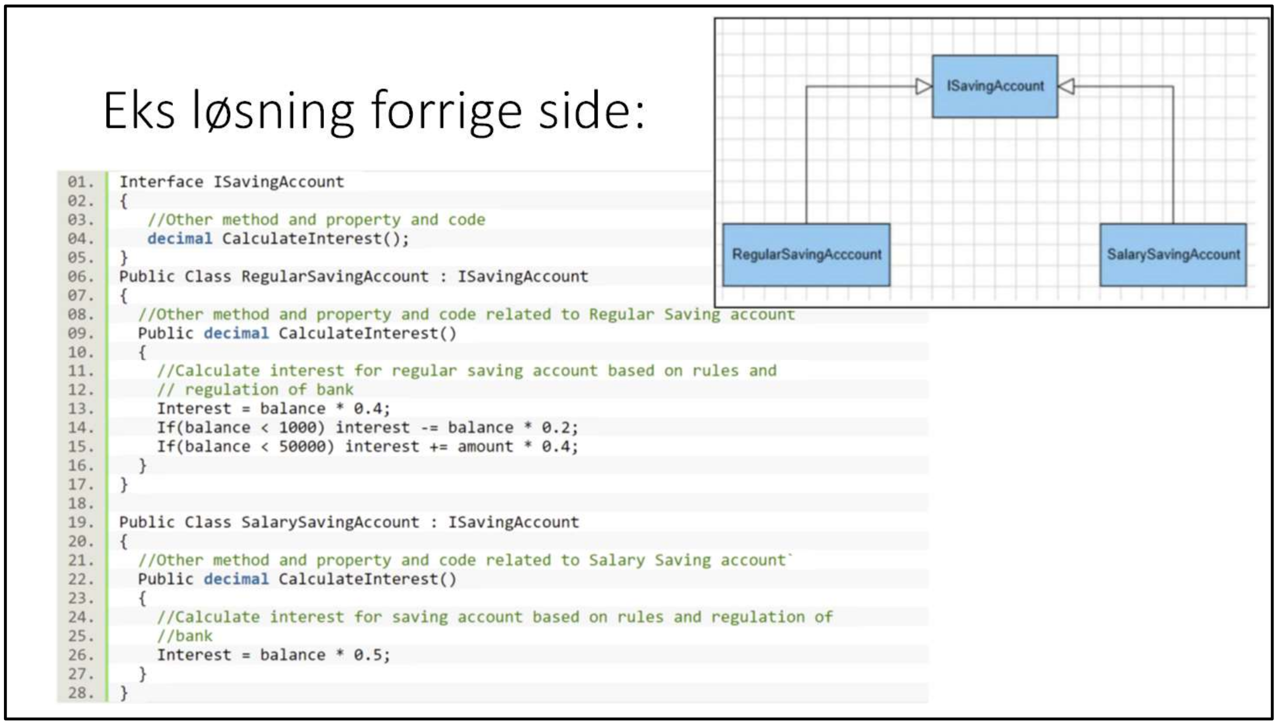
Principles

- Gitt følgende kode:
- Hvilket principle vill du brukt her?



```
Public class SavingAccount
{
    //Other method and property and code
    Public decimal CalculateInterest(AccountType accountType)
    {
        If(AccountType==”Regular”)
        {
            //Calculate interest for regular saving account based on rules and
            // regulation of bank
            Interest = balance * 0.4;
            If(balance < 1000) interest -= balance * 0.2;
            If(balance < 50000) interest += amount * 0.4;
        }
        else if(AccountType==”Salary”)
        {
            //Calculate interest for saving account based on rules and regulation of
            //bank
            Interest = balance * 0.5;
        }
    }
}
```

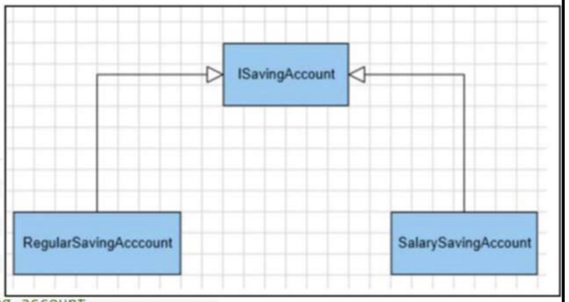
Open close principle – løsning neste side.



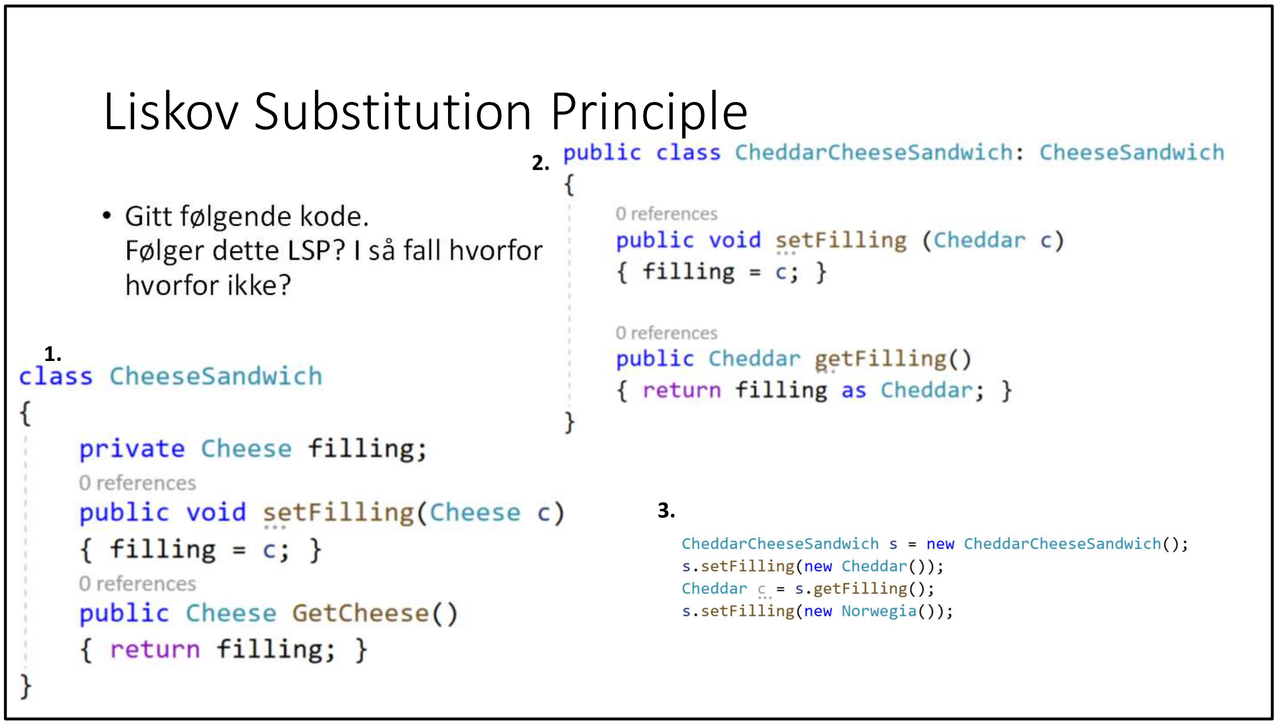
Eks løsning forrige side:

```
Interface ISavingAccount
{
   //Other method and property and code
   decimal CalculateInterest();
}
Public Class RegularSavingAccount : ISavingAccount
{
  //Other method and property and code related to Regular Saving account
  Public decimal CalculateInterest()
  {
    //Calculate interest for regular saving account based on rules and
    // regulation of bank
    Interest = balance * 0.4;
    If(balance < 1000) interest -= balance * 0.2;
    If(balance < 50000) interest += amount * 0.4;
  }
}

Public Class SalarySavingAccount : ISavingAccount
{
  //Other method and property and code related to Salary Saving account`
  Public decimal CalculateInterest()
  {
    //Calculate interest for saving account based on rules and regulation of
    //bank
    Interest = balance * 0.5;
  }
}
```



Ved bruk av open close principle



Liskov Substitution Principle

- Gitt følgende kode.
  Følger dette LSP? I så fall hvorfor hvorfor ikke?

1.

```
class CheeseSandwich
{
    private Cheese filling;
    0 references
    public void setFilling(Cheese c)
    { filling = c; }
    0 references
    public Cheese GetCheese()
    { return filling; }
}
```

2.

```
public class CheddarCheeseSandwich: CheeseSandwich
{
    0 references
    public void setFilling (Cheddar c)
    { filling = c; }

    0 references
    public Cheddar getFilling()
    { return filling as Cheddar; }
}
```

3.

```
CheddarCheeseSandwich s = new CheddarCheeseSandwich();
s.setFilling(new Cheddar());
Cheddar c = s.getFilling();
s.setFilling(new Norwegia());
```

Dette eks er gjennomgått i timer.
Nei, fordi CheddasCheeseSandwich.getFilling() ikke returnerer object av typen Cheese

Cohesion

- Hva er Cohesion, og hvorfor er dette et viktig principle inne software engineering?

- Cohesion.
  - module should not take on too many computation responsibilities
  - How to recognize a violation: If a class has many loosely or not-related attributes and methods
- From cohort – in the same group

Coupling

- Hva er coupling og hvorfor er dette viktig innenfor software engineering?
- Hvordan kan man håndtere en uønsket coupling?

- Dependency against implementation
- Many dependencies.
- Modules should not delegate responsibilities in many tiny parts
- How to recognize a violation: many outgoing links
- Better solution: Hierarchical delegation by limiting the number of dependencies for each delegate and letting the delegates further split the complex responsibilities

Software architecture

- What is a constraint?
- How is functional requirements related to architecture?

- A design decision where you have zero degree of freedom.
- Functionality does not determine architecture, there is no end to the architectures you could create to satisfy a given functionality
- Orthogonal

Use case

- Draw a use case diagram for the following situation:
  - A user can borrow a book from a library;
    - extend it with borrowing a journal
  - a user can give back a book to the library
    - including the use case when the user is identified

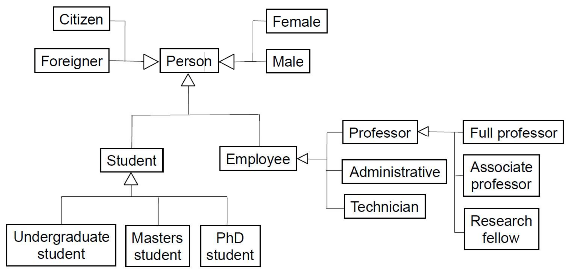
Use Case

- Main use cases: a customer buys something (eg. a book) from a virtual store like Amazon
- The user must be identified
- The book is not currently available, delayed delivery
- When the book is received the service must be graded
- The book is delivered via air mail
- The book is an ebook and can be delivered via Internet

- Hvilken av disse er extend og hvilken er include?

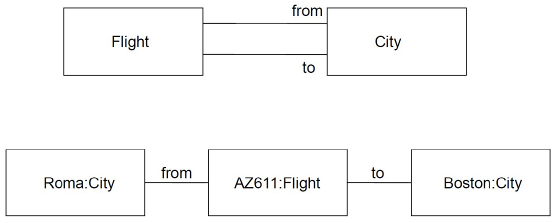
Domain model

- Utifra din forståelse av UiA, lag en domenemodell basert på følgende (se bort ifra behaviour foreløpig):
  - A university is an organization where some persons work, some other study
  - There are several types of roles and grouping entities
- Eks på løsning (gjennomgått i timen)



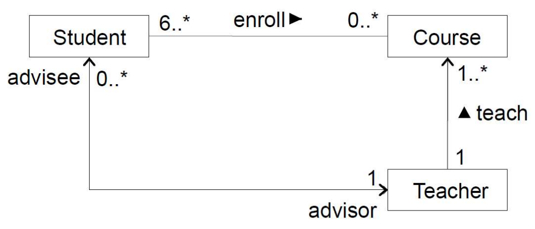
UML Classe diagram

- Vis ved hjelp av et klassediagram hvordan du vil fortelle en utvikler at en flyrute er assosiert med en by fra/til.
- Forslag ved hjelp av klassediagram og object diagram



UML

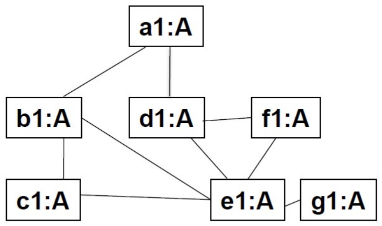
- Forklar følgende diagram:



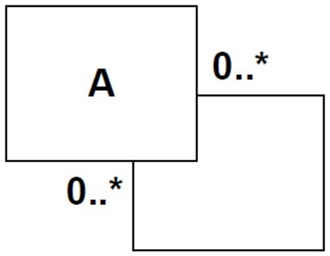
- En studen kan enroll i 0 til mange kurs.
- Et kurs må ha minimum 6 studenter, kan han ha *
- Et kurs undervises av 1 og bare 1 lærer.
- En lærer har minimum ett kurs,men kan ha mange.
- En student har 1 og bare 1 lærer som advisor.
- En lærer kan være advisee for 0 til mange studenter.

UML

- Forestill deg at en utvikler viser deg denne modellen. Hvordan ville du lage et klassediagram for denne.

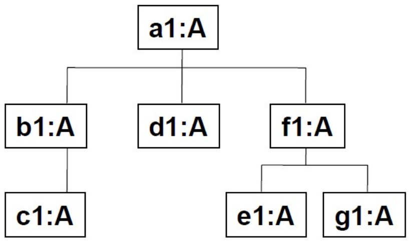


- Svar:

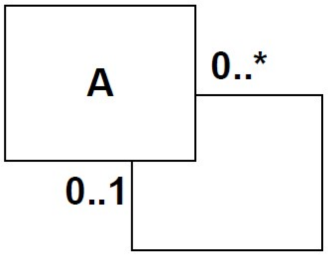


UML

- Forestill deg at en utvikler viser deg denne modellen. Hvordan ville du lage et klassediagram for denne.

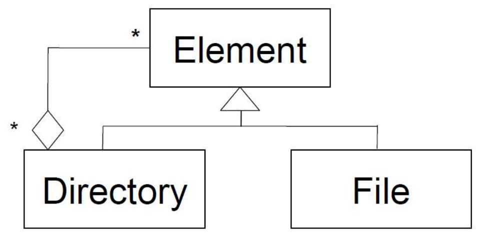


- Svar: den meste stricte ville være denne.



UML

- Forestill deg at du fikk denne beskrivelsen. Hvordan ville ditt klassediagram sett ut i UML?
  - A directory can contain any number of elements (either a file or a directory)
  - An element can be part of many directories
- Forslag (by the way, dette er ikke en lett oppgave. Oppgaven kan stilles motsatt vei, for en annen student, for å gjøre den lettere.)



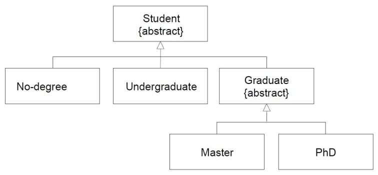
UML

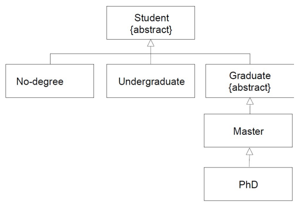
- Hvilken type association (Generalization, Association, Aggregation, Composition)er riktig hver tilfelle?
  - a) A country has a capital city
    b) A dining philosopher uses a fork
    c) A file is an ordinary file or a directory file
    d) Files contain records
    e) A class can have several attributes
    f) A relation can be association or generalization
    g) A polygon is composed of an ordered set of points
    h) A person uses a computer language on a project

- Svar:
  - a) A country has a capital city (C)
    b) A dining philosopher uses a fork (A)
    c) A file is an ordinary file or a directory file (G)
    d) Files contain records (C)
    e) A class can have several attributes (C)
    f) A relation can be association or generalization (G)
    g) A polygon is composed of an ordered set of points (AG eller C – kommer ann på hvordan man argumenterer)
    h) A person uses a computer language on a project (A)

UML

- Hva er forskjellen mellom de to modellene?

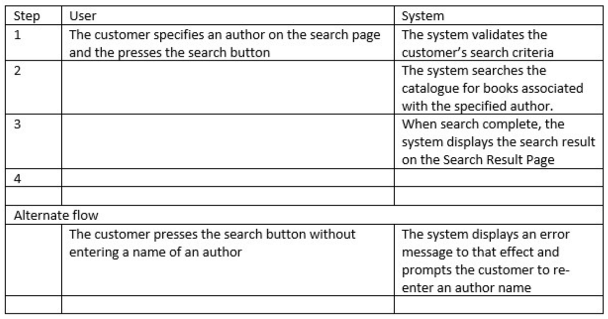




- Den øverste modellen forteller at Master og PhD er to uavhengige retninger.
- Den nederste modellen forteller at PhD er en forlengelse, påbygning av Master, og at man strengt tatt ikke an få en PhD uten å være Master.

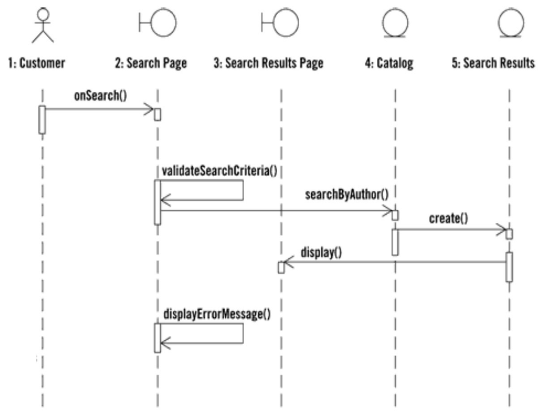
Use Case Description

- Tegn et interaction diagram basert på følgende.



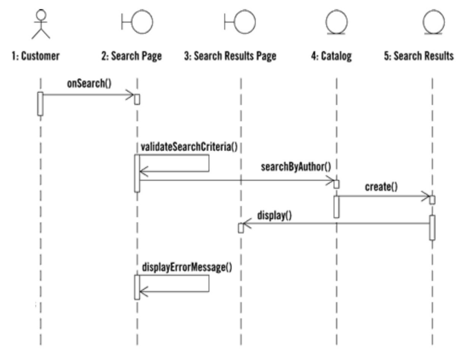
Step	User	System
1	The customer specifies an author on the search page and the presses the search button	The system validates the customer's search criteria
2		The system searches the catalogue for books associated with the specified author.
3		When search complete, the system displays the search result on the Search Result Page
4		
Alternate flow		
	The customer presses the search button without entering a name of an author	The system displays an error message to that effect and prompts the customer to re-enter an author name

- Eks for en forenklet flow



Interaction diagram

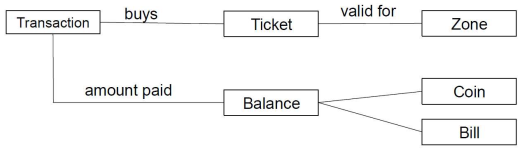
- Hva vil du si mangler i dette diagrammet basert på det vi har snakket om i timene?



- Svar: en controller klasse. Nå er det websiden som «driver» use casen, så systemet har nå mixet business logic i frontenden.

UML

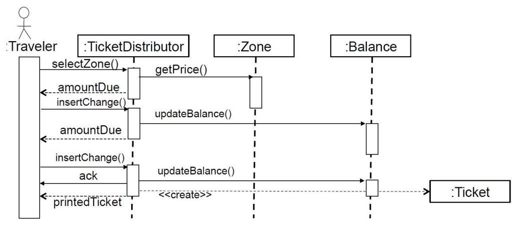
- Dette er en form for klassediagram, men hva ville du kalt dette hvis du skulle forklare meg det?



- Svar: domain model

Interaction diagram

- La oss si at du har følgende interaction diagram. Kan du gi en typebeskrivelse av klassene/objectene?



- Svar:
- TicketDistributor (TD) – tjener som både boundry og controller
- Zone: entity
- Balance: entity
- Ticket: entity
- Tilleggsspørsmål: solid principles spesielt ang TD, TD er sannsynligvis et SW system, og ikke en klasse.

Domain modelling

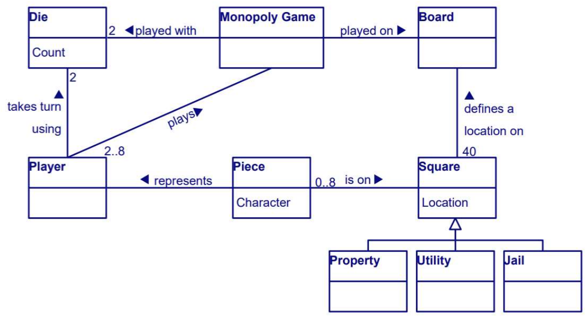
- Lag en domenemodel av følgende:
  - A customer interacts with a travel agency, a station and a train to reach some destination.

Interaction diagram

- Draw a sequence diagram to show how a user prints a document on a printer, and a counter keeps a count of printed pages

Monopoly example

- Lag en domene modell over slik du forstår spillet Monopol.



Sequence Diagram

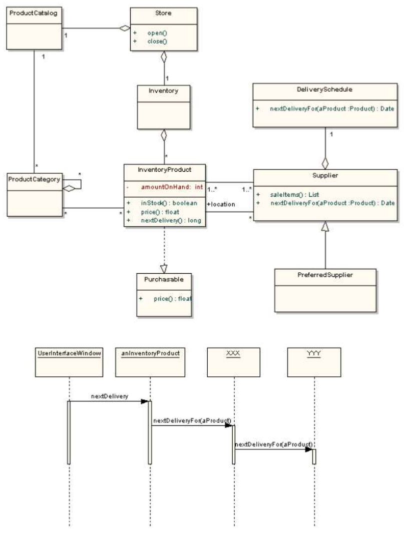
What are the names of missing classes XXX and YYY in the sequence diagram?

a) XXX = DeliverySchedule, YYY = Supplier

b) XXX = Supplier, YYY = DeliverySchedule

c) XXX = PreferredSupplier, YYY = DeliverySchedule

d) XXX = DeliverySchedule, YYY = PreferredSupplier

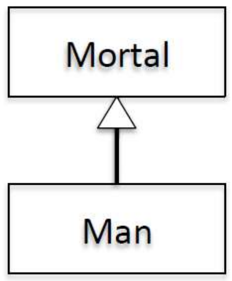


B og C er riktig.

UML diagrammer

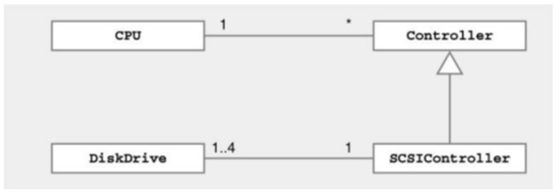
- Hvilket UML digram er best til å formidle følgende:
  «All men are mortal»
- A) Use case diagram
- B) Class diagram
- C) Interaction diagram
- D) Front end wire frame

- Svar: klassediagram



UML Computer System

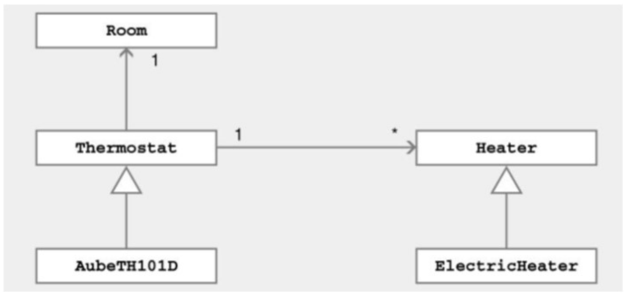
- Kan du forklare hva dette diagrammet ønsker å formidle?



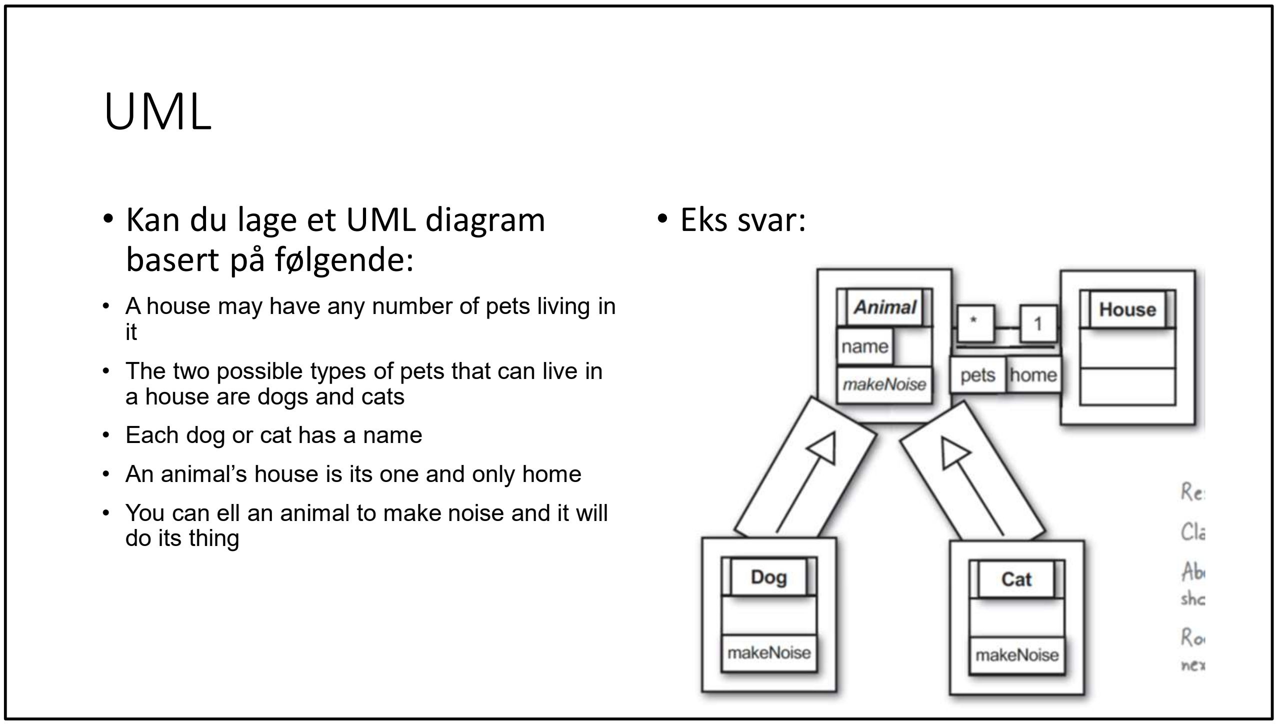
- Svar
  - 1 CPU associated with 0 or more Controllers
  - 1-4 DiskDrives associated with 1 SCSIController
  - SCSIController is a (specialized) Controller

UML Computer System

- Kan du forklare hva dette diagrammet ønsker å formidle?

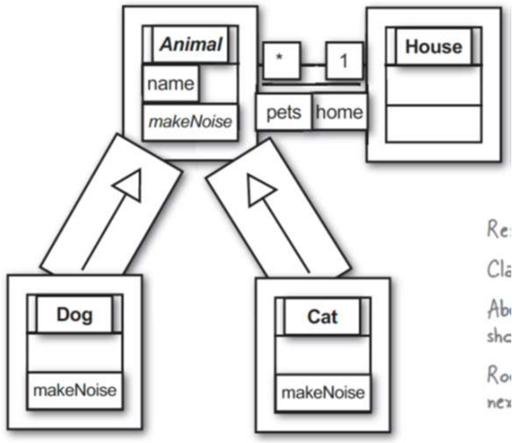


- Svar
  - Room has 1 Thermostat
  - Each Thermostat is associated with 0 or more Heaters
  - A Heather has exactly one Thermostat
  - ElectricHeater is a specialized Heater
  - AubeTH101D is a specialized Thermostat



UML

- Kan du lage et UML diagram basert på følgende:
  - A house may have any number of pets living in it
  - The two possible types of pets that can live in a house are dogs and cats
  - Each dog or cat has a name
  - An animal's house is its one and only home
  - You can ell an animal to make noise and it will do its thing
- Eks svar:



Endre den til kuddle, og få fram forskjellen mellom private og public variabel.