IN2000: Kravhåndtering, modellering, design og prinsipper

09 februar 2024

Yngve Lindsjørn

ynglin@ifi.uio.no

Kickoff – lørdag 17 februar fra 12.15 til 16

- Det vil bli praktisk informasjon for veien videre, og visning av tidligere apper som har blitt laget.
  - Besøk av tidligere studenter som forteller om sine erfaringer
- Presentasjon av oblig 3
- Pizza...
- Anledning for å bli bedre kjent med teammedlemmene og få en god start på prosjektarbeidet.
  - Diskutere case
- Påmelding her: https://nettskjema.no/a/399195

Forslag til case iår

– https://in2000.met.no/2024/

– Flere tips til APIer fra Meteorologisk og hvordan de brukes vil komme

Vinner-appen 2022

Ingen dårlig vær

Foran meg sitter tre av seks studenter som har laget appen Kleddy. De har laget en app som skal hjelpe små barn med å forstå hvordan de kler seg for været. Det er jo tross alt ingen dårlig vær, bare dårlig klær.

PROTOTYPE: Små barn kan øve på å kle seg etter været i appen Kleddy. (Foto: Kleddy)

– Vi fant veldig tidlig ut at saken vi valgte er ganske kompleks, forteller Eirik Langholm.

Konseptet til appen er enkelt nok at barn skal kle på en bamse og få visuelle tegn på om bamsen trives eller mistrives. Den kan for eksempel være for kald eller for varm.

Prosessen var krevende for gruppa. Ved brukertesting fant de ut av det planlagte antall knapper forvirret barna. Derfor har de tilpasset funksjonaliteten i appen for at barna skal klare å bruke den. Målgruppen til appen er barn mellom 4 og 7 år.

Veldig imponert

Værdataene kommer fra Meteorologisk Institutt, MET, som ligner i gangavstand fra Ifi. Roar Skålin er direktør ved MET og forteller at de er svært fornøyde med studentene og deres arbeid.

– En av grunnene til at vi syns dette er et godt fag, er at det gir studentene anledning til å jobbe med reelle data og reelle problemstillinger. Flere av appene de lager er interessante idemessig og kunne godt blitt utviklet til reelle apper, mener han.

" En av grunnene til at vi syns dette er et godt fag, er at det gir studentene anledning til å jobbe med reelle data og reelle problemstillinger. Flere av appene de lager er interessante idemessig og kunne godt blitt utviklet til reelle apper.

Roar Skålin, direktør ved MET

– Jeg er veldig imponert over hva de får til når de setter sammen ulike datatyper, fortsetter Skålin.

Tilbakemeldingene fra studentene er veldig gode å få for MET. Det er mange som bruker de åpne API-ene deres, men de færreste gir så gode tilbakemeldinger som studentene: hvordan API-ene fungerer og hvor enkelt det er å ta i bruk dataene i kombinasjon med andre data.

– Det er verdifull læring for oss, avslutter Skålin.

Vinnerapp 2021

Velkommen til

En app som gir deg alt annet enn bakglatte ski

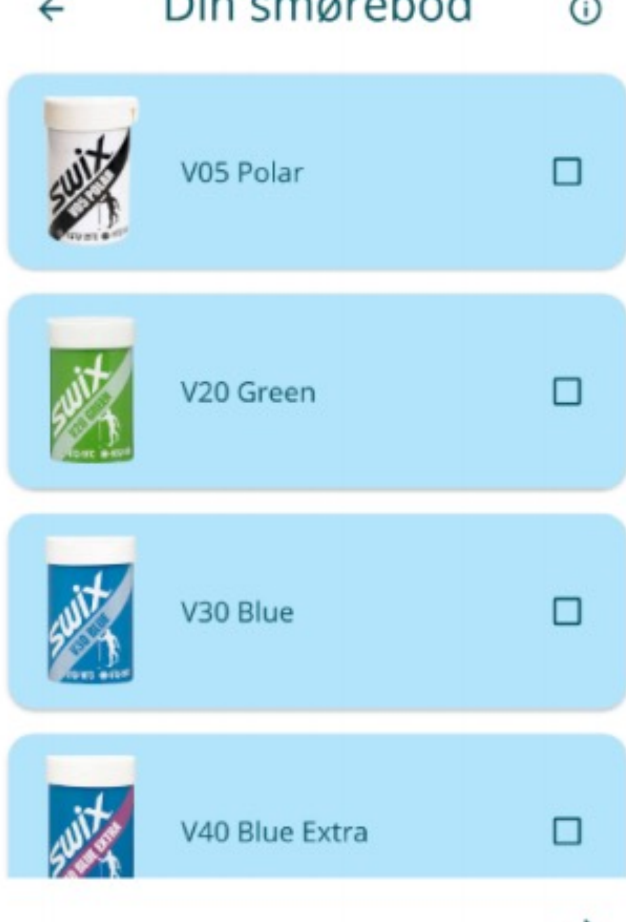
→

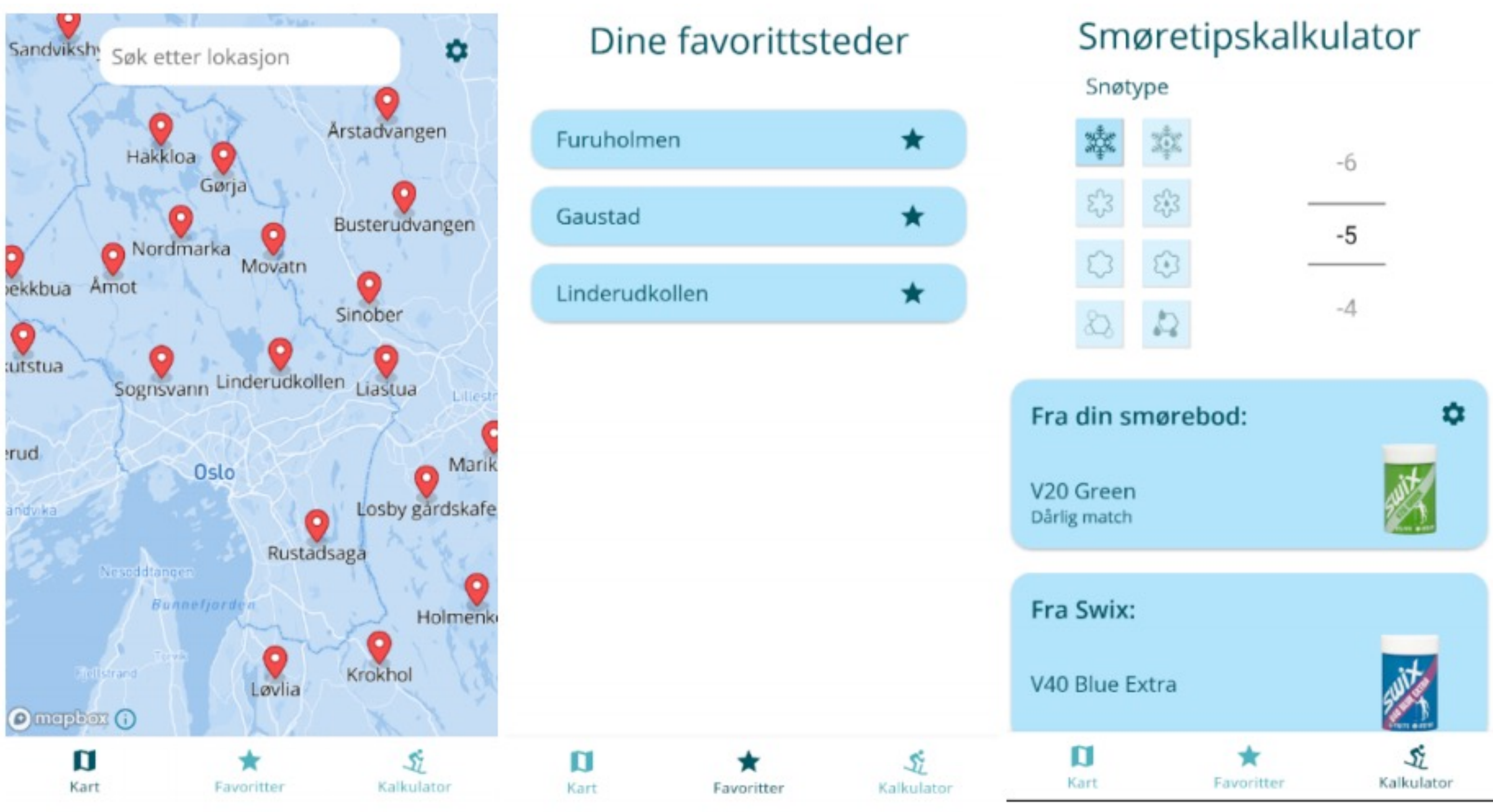
Vi ønsker å gi deg de beste smøretipsene ut fra den skismøringen du selv har

Legg gjerne til skismøring i din personlige smørebod på neste side

Dette kan endres i innstillinger senere

→





Søk etter lokasjon
Sandvikshy
Årstadvangen
Hakkloa
Gørja
Busterudvangen
Nordmarka
Movatn
Åmot
Sinober
Sognsvann
Linderudkollen
Liastua
Oslo
Losby gårdskafe
Rustadsaga
Krokhol
Løvlia
Dine favorittsteder
Furuholmen
Gaustad
Linderudkollen
Smøretipskalkulator
Snøtype
-6
-5
-4
Fra din smørebod:
V20 Green
Dårlig match
Fra Swix:
V40 Blue Extra
Kart
Favoritter
Kalkulator

Temaer

- Kravhåndtering
- Diagrammer
  - Usecase, - sekvens og klassediagrammer
- Design
  - Prinsipper og patterns

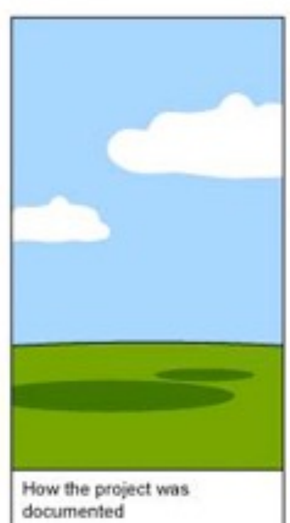
Gode beskrivelser av krav

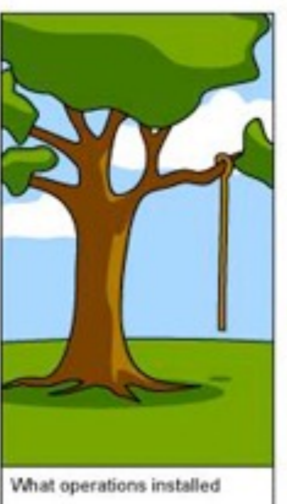
er viktig for

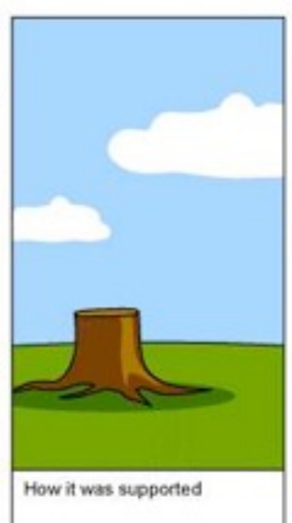
- kontrakt oppdragsgiver – leverandør
- planlegging og oppfølging
- arkitektur, design, test og sikkerhet
- å støtte videreutvikling og vedlikehold

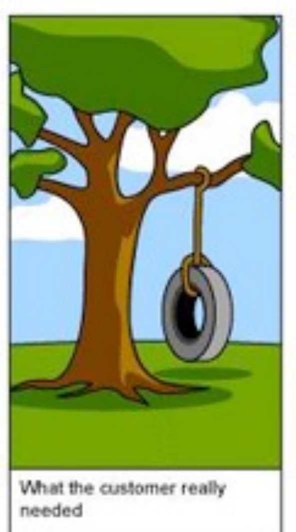
Utfordringer i kravhåndtering

- Kommunikasjon
- Felles forståelse







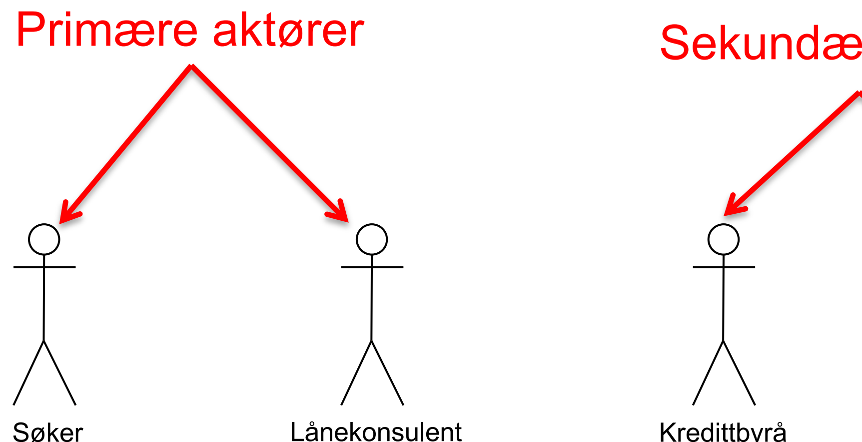


Beskrive krav

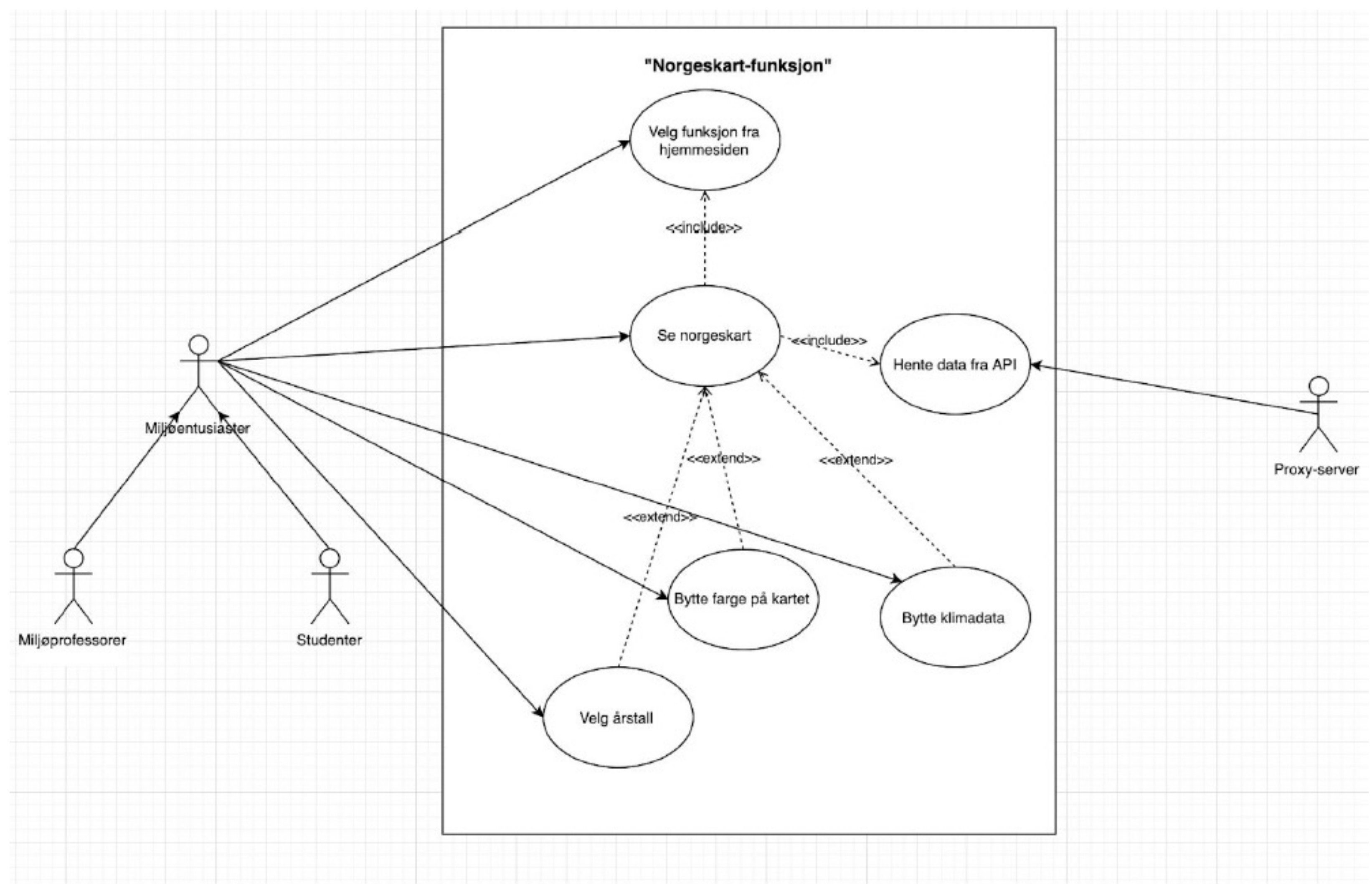
- Tekst
- Strukturert tekst
  - User story (brukerhistorie)
  - Use case (brukstilfelle)
- Modeller
  - UML (Unified Modeling Language)
  - BPMN (Business Process Model and Notation)

Use case modellering - Identifiser aktører

- **Primære aktører** har egne mål, dvs. de initierer use case (en eller flere) som oppfyller deres mål.
- **Sekundære aktører** har ikke egne mål, men er nødvendige for å realisere målene til de primære aktørene



Eksempel USE-case fra rapport i IN2000 – klima case

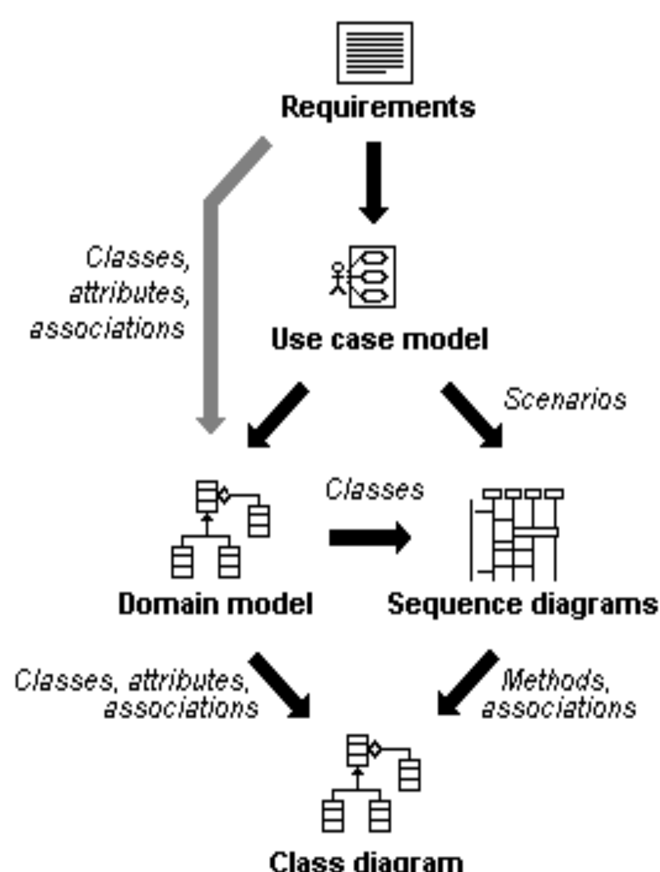


Use case i prosjektplanlegging

- Planlegg hvilke use case som skal implementeres i hvilke iterasjoner av prosjektet:
  - Implementer use casene i henhold til hvor viktige de er og/eller hvor vanskelige de antas å være å implementere.
  - Hovedflyt implementeres først, deretter alternativene.
  - Estimer hvor mange use case (eller hendelsesflyt og alternative flyt) som kan implementeres i en iterasjon.

Use case i design

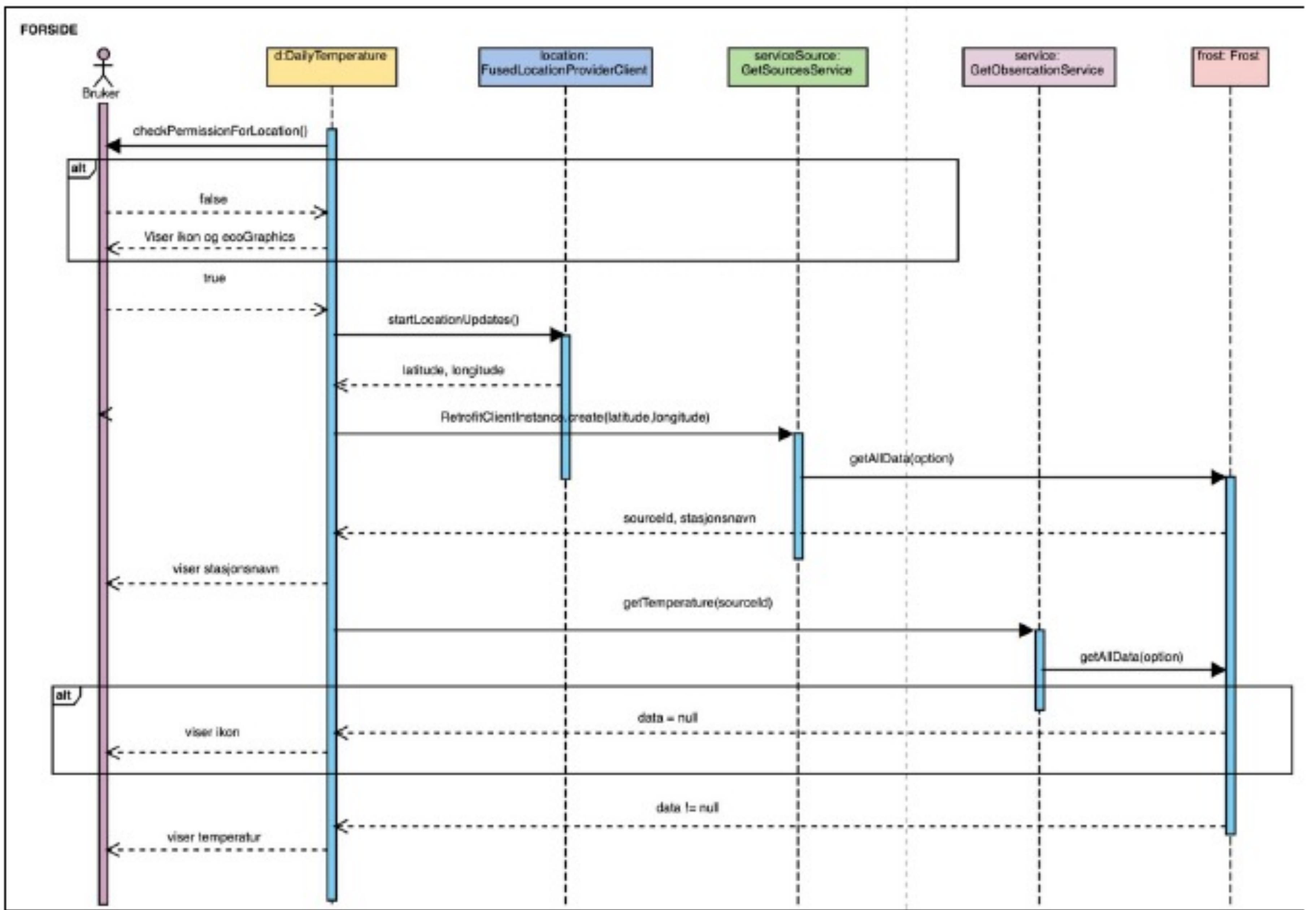
- Hendelsesflyt i use casene detaljeres ut i sekvensdiagram
- Domenemodellen utvides til klassediagram med systemklasser



Sekvensdiagrammer

- Et flyt i et use case kan modelleres med sekvensdiagrammer.
- For hvert use case lages typisk sekvensdiagram for hovedflyt og for hyppig forekommende alternative flyt.
- Stegene (sekvensene) i et use case vises som meldinger som sendes mellom objektene ved kall på objektenes metoder.

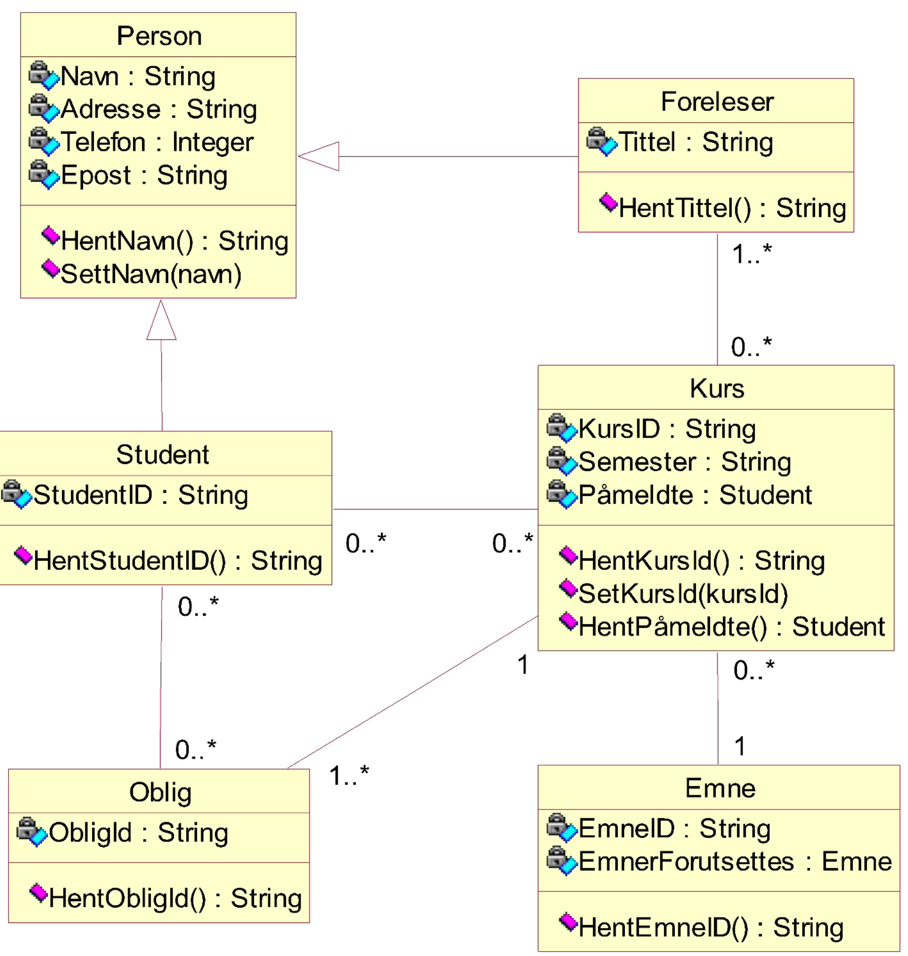
Eksempel klima case



Generalisering

- Det er ofte nyttig å undersøke klassene i et system for å se om det er mulighet for generalisering.
- I objektorienterte språk, er generalisering en del av språket – gjennom såkalt arvemekanisme (”inheritance”).
- Attributter og operasjoner (metoder) som er assosiert med ”superklasser” er også assosiert med ”subklasser” gjennom arv. Subklassene vil så legge til mer spesifikke attributter og operasjoner.

Klassediagram - Student tar kurs

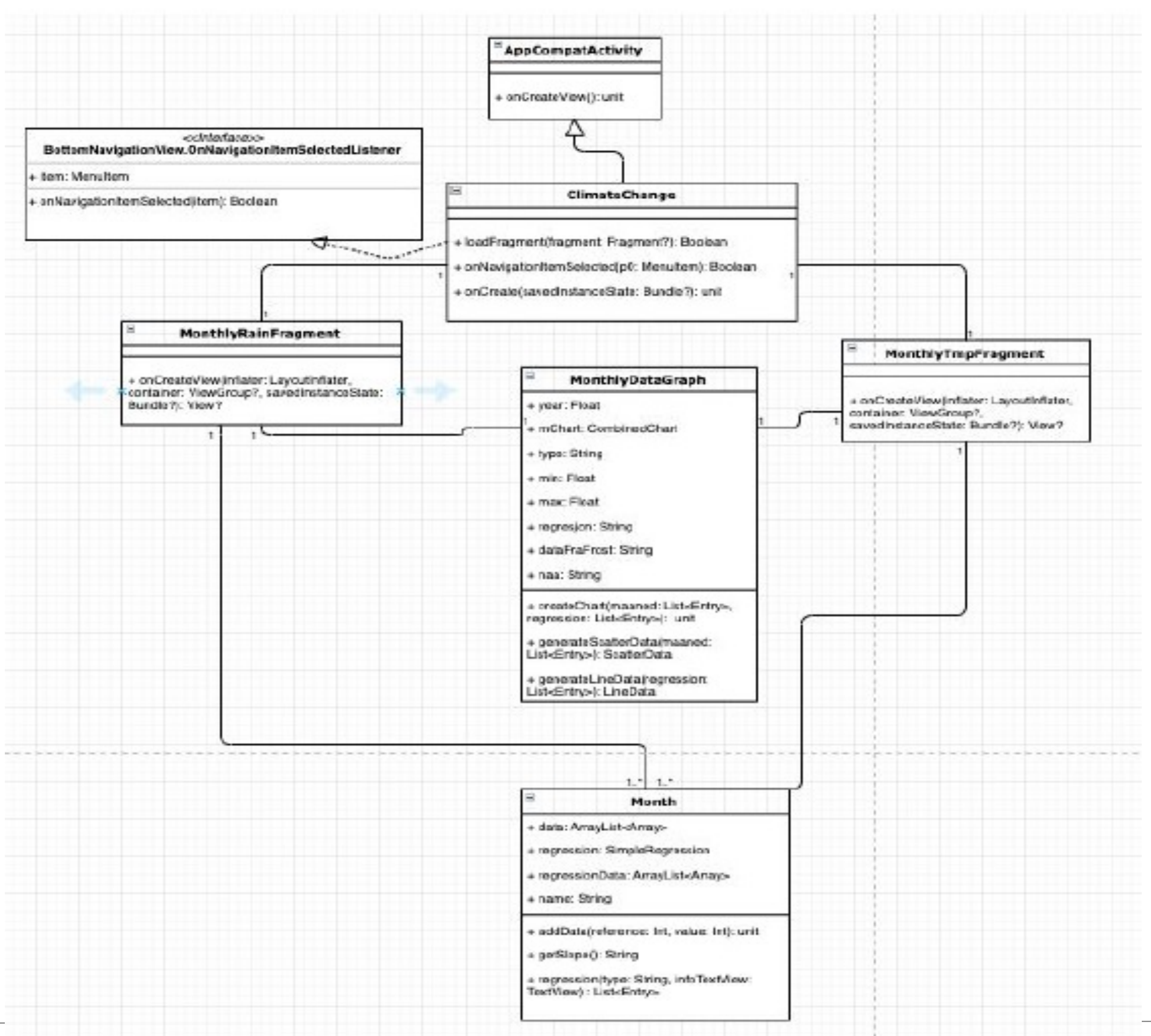


- Alle kurs har minst en obligatorisk oppgave.

Spesifikasjon av grensesnitt/interface

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

Eksempel klima case



Objektdesign: Ansvarstilordning

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

Kjennetegn på 'god' design

- En god utforming gjør den jobben den er ment å gjøre
- En god utforming er enkel og elegant
  - Eleganse innebærer å finne akkurat riktig abstraksjonsnivå
- En god utforming er gjenbrukbar, utvidbar og enkel å forstå
- Et godt objekt har et lite og veldefinert ansvarsområde
- Et godt objekt skjuler implementasjonsdetaljer fra andre objekter

- Grady Booch

Kvalitetsattributter
(se også lærebok tabell 4.2 og kapitel 8)

- Respons – Er responstiden god nok?
- Pålitelighet – Virker funksjonaliteten som forventet?
- Tilgjengelighet – Leveres funksjonaliteten når bruekrene ønsker det?
- Sikkerhet – Er systemet sikkert nok når det gjelder uautorisert inngrep eller angrep på systemet?
- Brukskvalitet – Kan brukerne navigere i systemet på en rask og feilfri måte?
- Vedlikeholdbarhet – Kan systemet videreutvikles og oppdateres med ny funksjonalitet uten for store kostnader?
- Robusthet – Kan systemet levere tjenester etter en større feil eller angrep?

Tips: Fra kapittel 8 i lærebok

- Bruk en programmeringsstil som reduserer sjansen for å introdusere feil
- Lag kode som er forståelig og lett å lese
- Unngå språkkonstruksjoner som lett blir «utsatt for feil»
- Valider input
  - Alder er et tall etc.
- Refaktorer koden – fjern uklar og uforståelig kode
- Gjenbruk velprøvd og sikker kode, og bruk velprøvde og konstruksjoner som er testet, slik som design patterns.

Figure 8.2 – Grunn til feil i programmer

- Problem – forstår ikke problemet eller problemområdet (domene)
- Teknologi – Programmeringsspråk, bibliotek, IDE, database
- Kompleks program - mange komponenter som virker sammen – kan være vanskelig å forstå hvordan tilstanden til programmet endres

Code smell

- **karakteristikker i kildekoden som kan indikere et større problem (se Tabell 8.6 i lærebok)**

- Store klasser – del opp i mindre klasser som er lettere å forstå (prinsippet om ett ansvarsområde – høy kohesjon)
- Store/lange metoder/funksjoner – Splitt lange metoder i mindre, mer spesifikke metoder og funksjoner. Gjør kun en ting.
- Duplisert kode – Må endre flere steder. Lag en singel instans av den dupliserte koden.
- Meningsløse navn på variable etc. – tyder ofte på hastverk. Erstatt med mer meningsfulle navn
- Ubrukt kode – reduserer lesbarheten av koden. Fjern denne koden

Høy kohesjon

- Kohesjon er et mål på hva slags ansvar et objekt har og hvor fokusert ansvaret er
- Et objekt som har moderat ansvar og utfører et begrenset antall oppgaver innenfor ett funksjonelt område har høy kohesjon
- Objekter med lav kohesjon har ansvar for mange oppgaver innen ulike funksjonelle områder

Lav kobling

- Kobling er et mål på hvor sterkt et objekt er knyttet til andre objekter
- Et objekt med sterk kobling er avhengig av mange andre objekter, noe som kan gjøre endring vanskelig

Designmønstre – ”Design patterns”

- Et designmønster er en måte å gjenbruke abstrakt kunnskap om et problem og løsningen på problemet
- Et mønster er en beskrivelse av et problem og essensen av løsningen
- Bør være tilstrekkelig abstrakt til å kunne bli gjenbrukt i ulike situasjoner
- Fra lærebok (kap 8.1.2): «A general reusable solution to a commonly occuring problem within a given context in software design»

Mer om Mønstre ("patterns")

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

Ekspertprinsippet: (Information Expert)

- **Problem**: Hva er det generelle prinsipp for å tilordne ansvar til objekter?
- **Løsning**: La det objektet som har kunnskapen (dataene) også behandle den
- **Hvordan**:
  - Begynn med å formulere ansvarsområdet:
  - Eks: Student-kurs:

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

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

Skaperprinsippet (Creator)

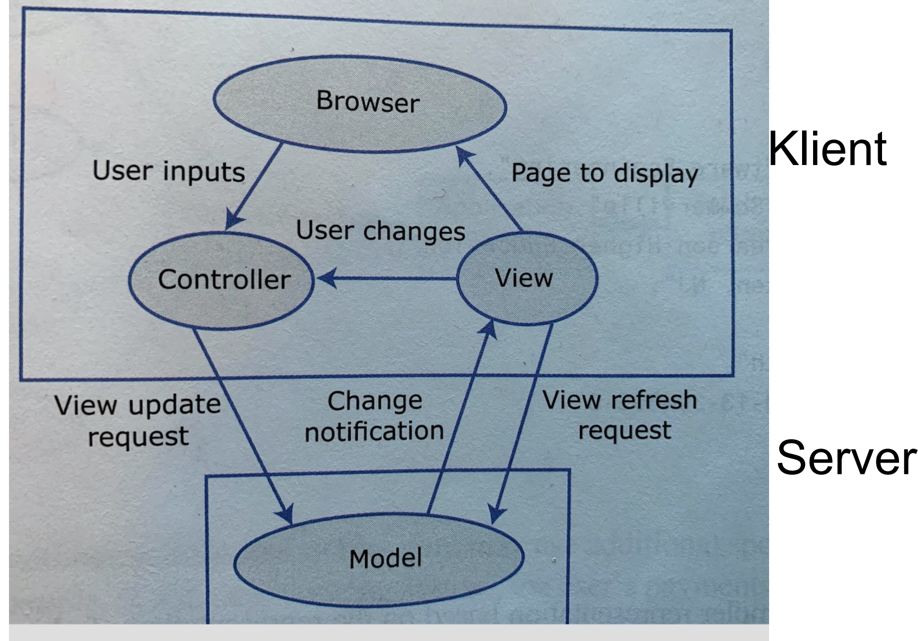
- Problem: Hvem er ansvarlig for å opprette nye objekter?
- Løsning: La det objektet som må vite om de nye objektene, lage dem
- Hvordan: Gi klasse B ansvaret for å opprette et objekt av klasse A dersom ett av følgende er sant:
  - B inneholder A-objekter
  - B registrerer A-objekter
  - B bruker A-objekter
  - B har data som sendes til A-objektet når det opprettes

Kontrollobjektprinsippet (Controller)

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

MVC - Model View Controller

- Klient – server applikasjoner bruker ofte MVC pattern
- Klienten oppdateres når data endres på serveren
- «model» representerer system data og «business» logikk
  –
- Alle «views» blir oppdatert når modellen endres



Klient

Server

Fra lærebok figur 4.13

Eksempler på tidligere oppgaver IN2000

2018:

- Forklar arkitekturen bak MVP (Model View Presenter) og redegjør for hvordan dere brukte det i prosjektet?
- Forklar prinsippene høy kohesjon og lav kobling. Forklar med eksempler hvordan dere brukte dette i prosjektet?

2019:

Gjør rede for arkitektur/patterns dere brukte i prosjektarbeidet, og hvordan patterns kan hjelpe med å unngå teknisk gjeld. Dersom dere ikke fulgte et bestemt pattern, gjør rede for hvordan dere kunne ha brukt dette for å unngå teknisk gjeld.

2020:

a) Hvorfor tror du det vanlig å benytte seg av design patterns innen programvareutvikling?
b) Redegjør for hvordan design pattern ble realisert i deres prosjekt, og hvordan dette påvirket arkitekturen i appen. Om dere ikke benyttet dere av noe design pattern, forklar hvorfor dere ikke gjorde det og hvordan arkitekturen kunne blitt annerledes dersom dere hadde benyttet det.

2021:

Du skal videre ta for deg use-caset "book badstue". Redegjør for eventuelle antagelser i besvarelsen for de følgende oppgavene.

a) Lag et aktivitetsdiagram for use-caset "book badstue".
b) Lag et sekvensdiagram for use-caset "book badstue". Modeller hovedflyt og en alternativ flyt.
c) Lag et klassediagram som reflekterer sekvensdiagrammet laget i oppgave b)

Husk å få med assosiasjoner og tilhørende multiplisitet mellom klassene.

2022:

Nevn 5 eksempler på karakteristikker i kildekode som kan indikere et problem (såkalt «code smells») og forklar hva som kan gjøres for å unngå dem.