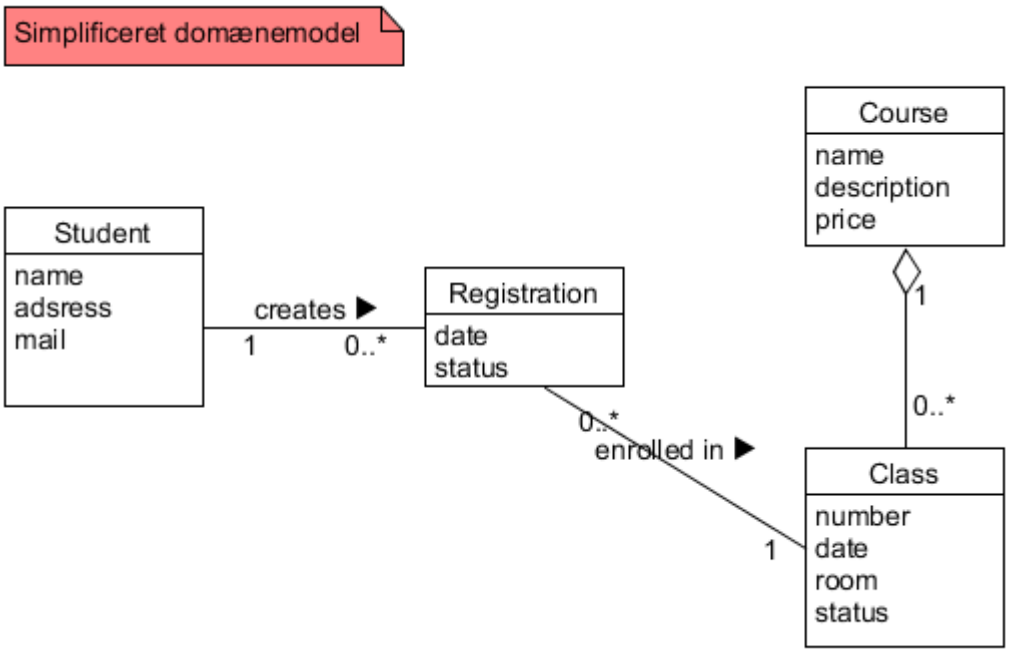
Systemudvikling – Lektion 6

Opgaver i design

Opgave 1 interaktionsdiagram og designklassediagram

Vi fortsætter med kursustilmeldingssystemet, som I har arbejdet med. Her er et forslag til en simplificeret domænemodel:

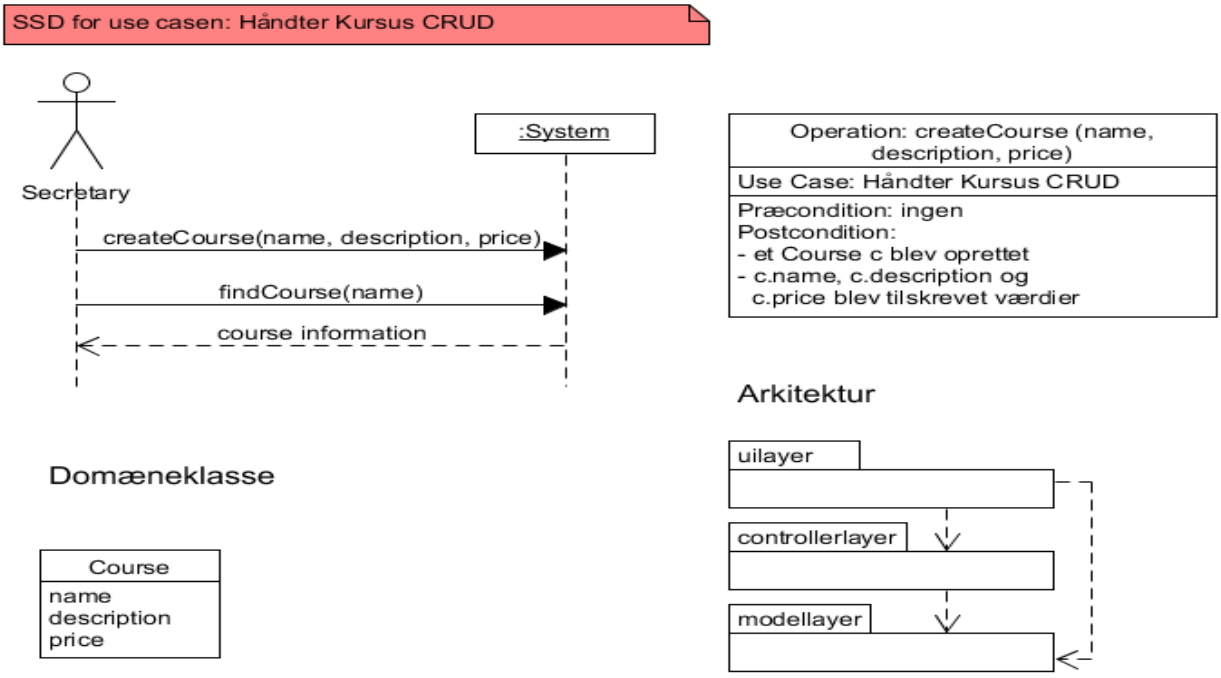


Baseret på denne domænemodel og informationen i dagens præsentation + Larman, skal I lave interaktionsdiagrammer (kommunikation- eller sekvensdiagrammer) for `createStudent(name, address, mail)`, der er en del af use casen håndter kursist CRUD.

1. start med designklassediagrammet og tilføj de klasser der vil give mening i forhold til CRUD for kursister (student på diagrammet). Husk den tredelte arkitektur og at vi her på 1. semester bruger containerklasser til at håndtere modellaget med.
2. lav et interaktionsdiagram for createStudent, I bestemmer selv om I vil lave det som kommunikationsdiagram eller som sekvensdiagram. Viser det sig at der mangler metoder eller klasser vend da tilbage til designklassediagrammet og tilføj disse.
3. Baseret på det udleverede BlueJ projekt skal I nu kode diagrammet. Der er vedlagt et skelet projekt som har en meget basal UI. Dette skal I bruge som basis.

Opgave 2 createCourse design og kode

SSD og kontrakter for *Håndter Kursus -**CR**UD* er vist i nedenstående figur:



1. lav interaktionsdiagram for `createCourse(name, description, price)`. Jeg vil anbefale at i bruger den modsatte diagramstil end den I brugte i opgave 1 for at få erfaring med begge stile således at I kan træffe et oplyst valg af diagramstil.
2. så skal I udvidde designklassediagrammet med de nye klasser, metoder, atributter og forbindelser.
3. endelig skal koden udviddes med den nye funktionalitet. Der er et skelet tilgængeligt i dagens modul

Opgave 3 FindStudent design og kode

Nu skal findStudent designes og implementeres. Den første beslutning der skal træffes er om man skal søge på id eller på navn. Hændelsen går fra StudentUI til StudentController, hvorfra den sendes videre til StudentController. Heri laves en søgning på enten navn eller id afhængig af jeres design beslutning

1. start med at lave interaktionsdiagram for `findStudent(…)`. Benyt samme diagrammeringsstil som i opgave 1
2. dernæst udviddes designklassediagrammet med nye klasser, metoder, atributter og forbindelser (hvis nødvendigt)

3. Koden skal udvides med den nye funktionalitet. Husk at teste det af. Metoder, atributter og klasser skal have de samme navne som i diagrammerne.

Opgave 4 findCourse design og kode

I skal nu designe og kode systemhændelsen findCourse(name). Hændelsen går fra CourseUI til CourseController, der sender den til CourseContainer. Deri laves en søgning på navn. Det er nu op til jer at bestemme om der skal returneres en liste med matchende navne eller kun det første kursus med navne match.

1. start med at lave interaktionsdiagram, brug den samme stil som I valgte i opgave 2
2. udvid designklassediagrammet så det passer med de nye metoder, klasser, attributter og forbindelser (hvis der er nogle). Som altid vil man lave lidt på interaktionsdiagram så på designklassediagram osv.
3. Koden skal udvides med den nye funktionalitet. Husk at teste. Navngivningen skal passe 100% med hvad diagrammerne siger, ellers har I fejl i opgaven.

Opgave 5: Færdiggør analysen og designet af use casen: *registrer Afholdelse* og kod derefter designet

Use casen kunne se således ud:

Use case: *Registrer afholdelse*

Aktør: Sekretær

Præbetingelse: Kurset findes

Postbetingelse: En ny afholdelsen er registreret og tilføjet et kursus

Normal forløb

1. Der er planlagt nye afholdelser af et kursus som er klar til at blive registreret med henblik på tilmelding
2. Sekretæren starter med at søge kurset ved angivelse af navn
3. Systemet returnerer kursusoplysninger
4. Sekretæren inddaterer oplysninger om den nye afholdelse (hold)
5. Systemet registrerer oplysningerne

Step 4 og 5 gentages indtil alle nye afholdelser er registreret

1. lav SSD og kontrakter (medmindre I allerede har disse)
2. Dernæst designes use casen
    a. Kig på designklassediagrammet I allerede er startet på og tilføj evt. nye klasser, som er nødvendige for at kunne realisere interaktionen i denne use case. Da der er en aggregering mellem klasserne *Course* og *Class* bør listen med samlingen af *Class* placeres på det *Course* de hører til, dvs *Course* er container for *Class*.
    b. Udarbejd et interaktionsdiagram
3. Udvid designklassediagrammet med de relevante klasser, associeringer, navigering, synligheder og metoder (Husk trelagsarkitekturen)
4. Overvej synligheden mellem *Class* og Course, så man fra en Class kan se hvilket Course den er tilknyttet
5. Optional Hvis I er foran og diagrammrne er på plads kan I kode designet i blueJ.

Opgave 6: Design af *FindClass(number)*

Design systemhændelsen/operationen: *FindClass(number),* som I skal bruge i næste opgave. Kom med forslag til hvordan dette skal designes ved først at kigge på hvilke klasser der skal indgå heri, lave kommunikationsdiagrammet og til sidst udvide designklassediagrammet som beskrevet i ovenstående.

I designet skal I overveje følgende:

- Først at iterere gennem *CourseContaineren.* For hver instans af *Course* skal I dernæst iterere over de afholdelser, der er på kurset (class[i]). Når der er match har I fundet afholdelsen(class). Så der kan med fordel anvendes nested loops.

Opgave 7: Lav use case fully dressed beskrivelse (hvis I ikke allerede har gjort det), SSD og operationskontrakter for use casen: *Tilmeld Kursist*

Brief beskrivelsen kunne se således ud:

Use case: *Tilmeld kursist*

En kursist har kigget i kursuskataloget og ringer ind for at tilmelde sig et hold (kursusafholdelse) han har fundet. Sekretæren bruger systemet til at finde den ønskede afholdelse, registrere kursisten og lave tilmeldingen.

Opgave 8: Påbegynd designet af use casen: *Tilmeld Kursist*

1. I har designet systemhændelsen: *findClass (number)*. Da use casen Tilmeld Kursist indeholder et skridt hvor man finder den relevante afholdelse (class) skal I overveje at genbruge *CourseController* og relaterer *RegistrationController* (eller hvad I nu ender op med at kalde controlleren) hertil. Derved understøttes princippet om lav kobling
2. *createStudent* er en simpel CRUD aflæsning som I designede og kodede i opgave 1. Her skal I overveje genbrug af *StudentController* for at understøtte princippet om lav kobling.
3. Nu kender I både de relevante instanser af afholdelse (class) og kursist(student) og skal nu koble disse instanser/objekter sammen i *createRegistration* jf operationskontrakten
4. Ud fra interaktionsdiagrammet udvider I herefter designklassediagrammet med relevante klasser, datatyper, metoder og synligheder

Opgave 9 Refaktorering af kode (optional)

En del af arbejdet når man udvikler og koder et system er at opdage kodeduplikation. Dette findes i det udleverede kodeskelet (i uilayer). Hvis du føler for det kan du refaktorere koden til UI således at den duplikation der er på tværs af de enkelte UI klasser forsvinder. Her tænkes på de metoder der prompter for et input. Se evt. Zuul opgaverne i BlueJ bogen for inspiration til en løsning.