Systemudvikling – Modul 7

Opgave 1: 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 2: 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 3: 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 4: Design 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

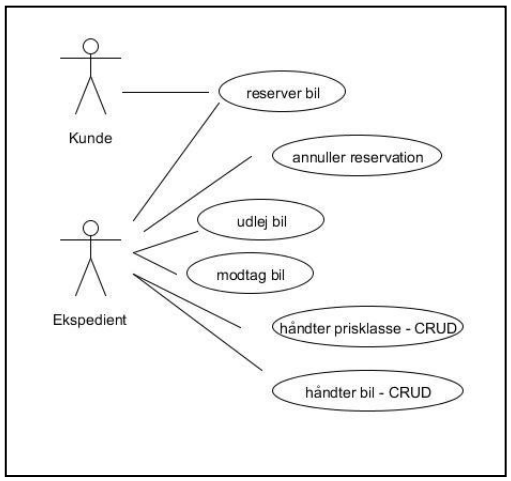
4. Ud fra interaktionsdiagrammet udvider I herefter designklassediagrammet med relevante klasser, datatyper, metoder og synligheder

Opgave 5 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.

Opgave 6: Kajs Biler

I de første moduler arbejdede I på Kajs Biler. I lavede bl.a. et use case diagram, beskrev dem brief og de højst prioriterede fully dressed. Her er et eksempel på use case diagram og beskrivelse af use casen: *Reserver bil* fully dressed



Use-case: Reserver bil

Aktør : Ekspedient

Præbetingelse: Kunden er ikke kendt. Ønsket prisklasse med tilhørende biler er registreret

Post betingelse: Reservationen er gemt, og tilføjet kunde- og prisklasseoplysninger

Flow of events

1. Use-casen starter med at en kunde henvender sig for at reservere en bil
2. Ekspedienter angiver ønsket prisklasse og periode
3. Systemet tjekker om der er ledige biler
4. Ekspedienten angiver kundeoplysninger
5. Systemet registrerer kunden
6. Ekspedienten afslutter reservationen
7. Systemet gemmer reservationen

1. Lav domænemodellen
2. Lav SSD og operationekontrakter for use casen: *Reserver bil*
3. Design use casen: *Reserver bil*
    a. Find først de klasser I skal bruge
    b. Design interaktionen mellem objekterne i et kommunikationsdiagram. Husk GRASP principper og Singleton
4. Lav designklassediagrammet. Påfør:
    a. klasser, attributter og datatyper
    b. metoder
    c. navigering og synligheder (referenceattributter)
5. Har I tiden til det må I gerne prøve at kode dele af designet