GruppeD | dm75

TEMADESIGN

Morten Løvborg Jensen
Kristian Vassard
Mikael Lehmann Kristensen

Forord

Opgaven for gruppen er, at udvikle en applikation med en adressebog, der indeholder oplysninger om venner, samt oplysninger om en DVD samling, hvorfra det skal være muligt at udlåne DVD'er til disse venner.

Adressebogen skal indeholde metoder til at tilføje venner, aflæse oplysninger, opdatere oplysninger samt slette venner, altså en CRUD (Create, Read, Update og Delete).

DVD'erne skal gemmes i en samling, hvori det skal være muligt at tilføje, læse, opdatere og slette DVD'er fra, igen en CRUD.

Ved udlån af DVD'er skal det være muligt at registrere hvem der er låneren, samt hvilke og hvor mange eksemplarer låneren har lånt, desuden skal det registreres når låneren afleverer det lånte.

Formålet med opgaven er, at opnå en grundlæggende forståelse for aktiviteter og produkter i design og programmering.

Indholdsfortegnelse

1. Indledning

I forbindelse med opgaveløsningen, aftalte vi i gruppen at:

- Møde til tiden.
- Overholde aftalte opgaver.
- Være engageret i arbejdet.
- Giv plads til alles meninger.

Samtidig blev vi enige om følgende kodestandard:

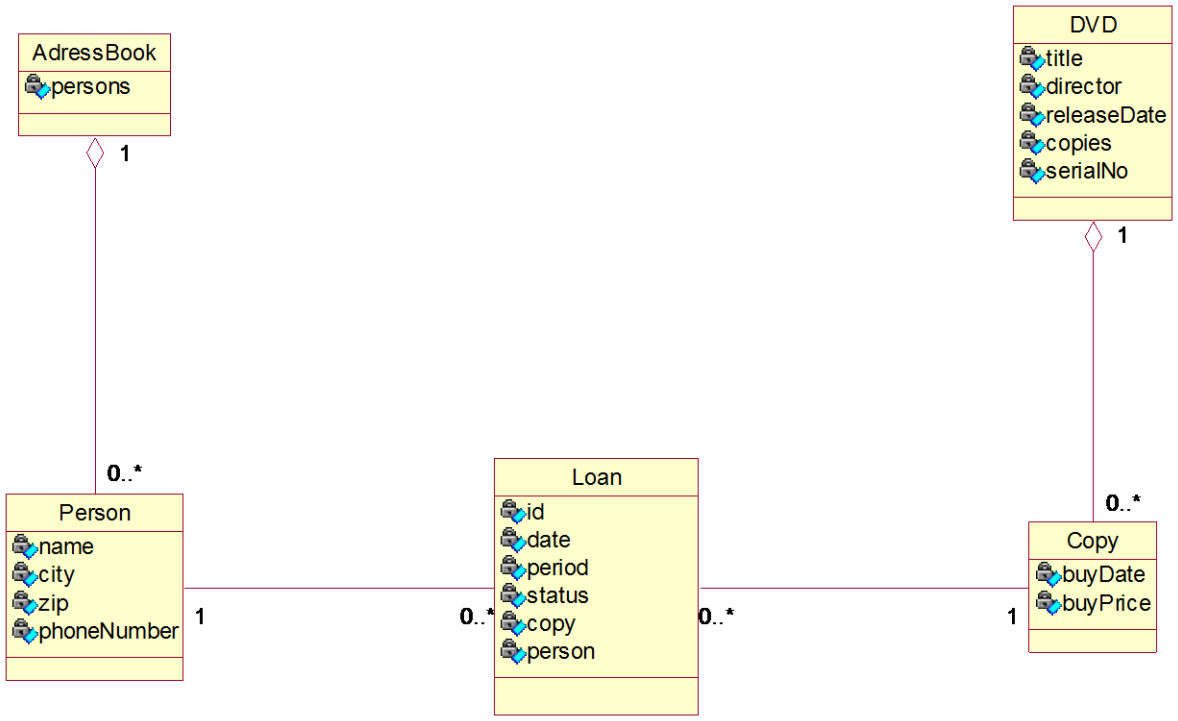
- Selve koden skal foregå på engelsk, klasser, variabelnavne, metoder, kommentarer osv.
- Lav kobling og høj samhørighed.
- Sigende variabel- og metodenavne.
- Turborg altid på ny linje
- Indrykning som standard.
- Alt koden skal testes inden commit.

2. Opgaven

I tilknytning til forordet, hvor opgaven for gruppen kort står beskrevet, vil vi i følgende kapitel vise informationen (domænemodel) og funktionaliteten (use case diagram) for den applikation som gruppen skal udvikle.

Yderligere identificerer vi hvilke use cases systemet indeholder, og disse bliver herefter beskrevet i de følgende kapitler.

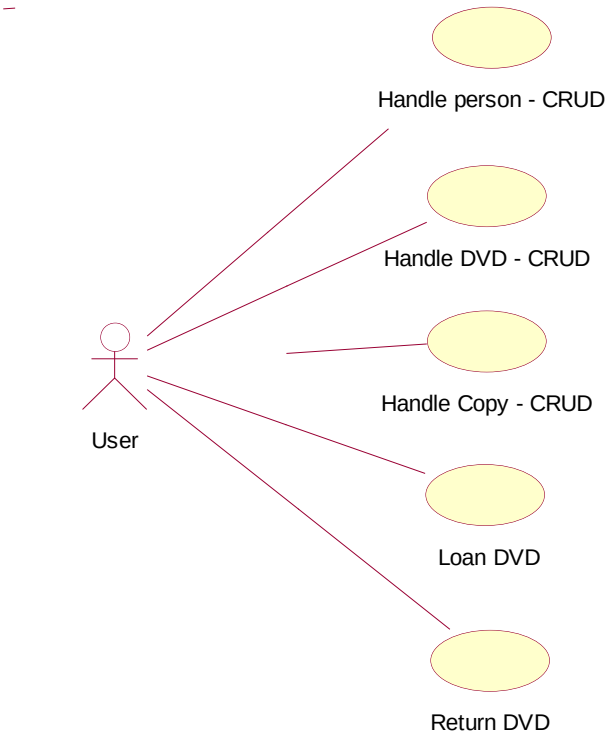
2.1 Domænemodel



Ovenstående UML klassediagram viser associeringen imellem et udlån af et dvd-eksemplar og et person objekt. Yderligere er der en aggrigering af det overordnede Person objekt (helheden) bestående af et antal person objekter (delene) i AdressBook.
Dette er samtidig et godt eksempel på genstands-beskrivelsesmønsteret, da ethvert DVD objekt har en beskrivelse (beskrivelse) som fastlægger det generelle, som er fælles for alle copy objekterne (genstandende) – det enkelte copy objekt fastlægger så de konkrette forhold for dette objekt.

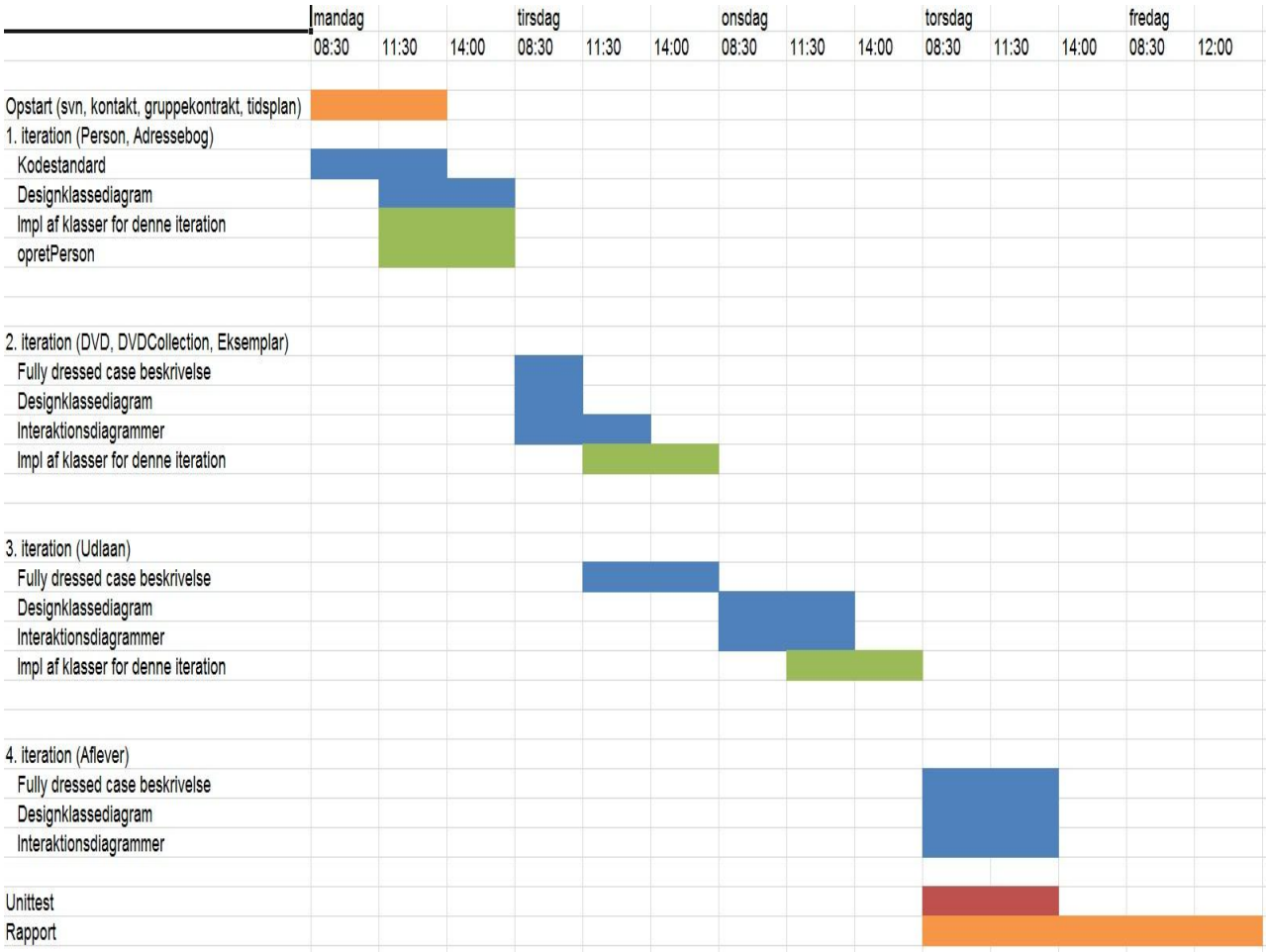
Dette mønster er særligt nyttigt i systemer, som skal administrere forskellige slags beskrivelser såsom vores produktspecifikationer for de enkelte DVD'er.

2.2 Use case diagram



Ovenstående use case diagram fastlægger systemets funktionalitet og viser aktøren, som er brugeren af programmet, og de use cases som vores system indeholder – i alt sin enkelthed viser diagrammet her interaktionen mellem aktøren og systemet.

3. Tidsplan


mandag
tirsdag
onsdag
torsdag
fredag
08:30
11:30
14:00
08:30
11:30
14:00
08:30
11:30
14:00
08:30
11:30
14:00
08:30
12:00
Opstart (svn, kontakt, gruppekontrakt, tidsplan)
1. iteration (Person, Adressebog)
Kodestandard
Designklassediagram
Impl af klasser for denne iteration
opretPerson
2. iteration (DVD, DVDCollection, Eksemplar)
Fully dressed case beskrivelse
Designklassediagram
Interaktionsdiagrammer
Impl af klasser for denne iteration
3. iteration (Udlaan)
Fully dressed case beskrivelse
Designklassediagram
Interaktionsdiagrammer
Impl af klasser for denne iteration
4. iteration (Aflever)
Fully dressed case beskrivelse
Designklassediagram
Interaktionsdiagrammer
Unittest
Rapport

3. Iteration 1

I følgende kapitel beskrives use casen ”Handle person - CRUD”, og ud fra denne skrives SSD med systemhændelser og tilhørende operationskontrakter. Ud fra use casen og SSD’en udarbejdes interaktionsdagram, hvorefter vores design klasse diagram opdateres.

3.1 Use case

createPerson ”C”
Primær aktør: User
Pre-betingelse: En collection AddressBook er oprettet
Post-betingelse: Et Person objekt er oprettet og associeret med AddressBook
Hoves succes scenarie:

- Brugeren angiver Person oplysninger (name, address, zip, phoneNumber)
- Systemet accepterer og opretter et Person objekt

findPerson ”R”
Primær aktør: User
Pre-betingelse: Person objekt er oprettet
Post-betingelse: Person objekt er fundet
Hoves succes scenarie:

- Brugeren angiver det rigtige person.id
- Systemet accepterer oplysningerne og finder det søgte person objekt
- System returnerer det søgte person objekt

updatePerson ”U”
Primær aktør: User
Pre-betingelse: Person objekt er fundet
Post-betingelse: Systemet opdaterer person objekt
Hoves succes scenarie:

- Brugeren ændrer person objektets oplysninger i systemet
- Systemet accepterer oplysningerne og opdaterer person objektet

deletePerson ”D”
Primær aktør: User
Pre-betingelse: Person objekt er fundet i systemet
Post-betingelse: Systemet har slettet Person objekt
Hoves succes scenarie:

- Brugeren sletter person objektet i systemet
- System accepterer og sletter person objektet

Ovenstående use case er skrevet som CRUD, og giver anledning til følgende SSD.

3.2 System Sekvensdiagram og operationskontrakter

SSD

SSD er et forhandlingsforløb som viser aktøren, systemet og de systemhændelser/input aktøren genererer i forhold til systemet, deres rækkefølge, samt systemets respons på dette.

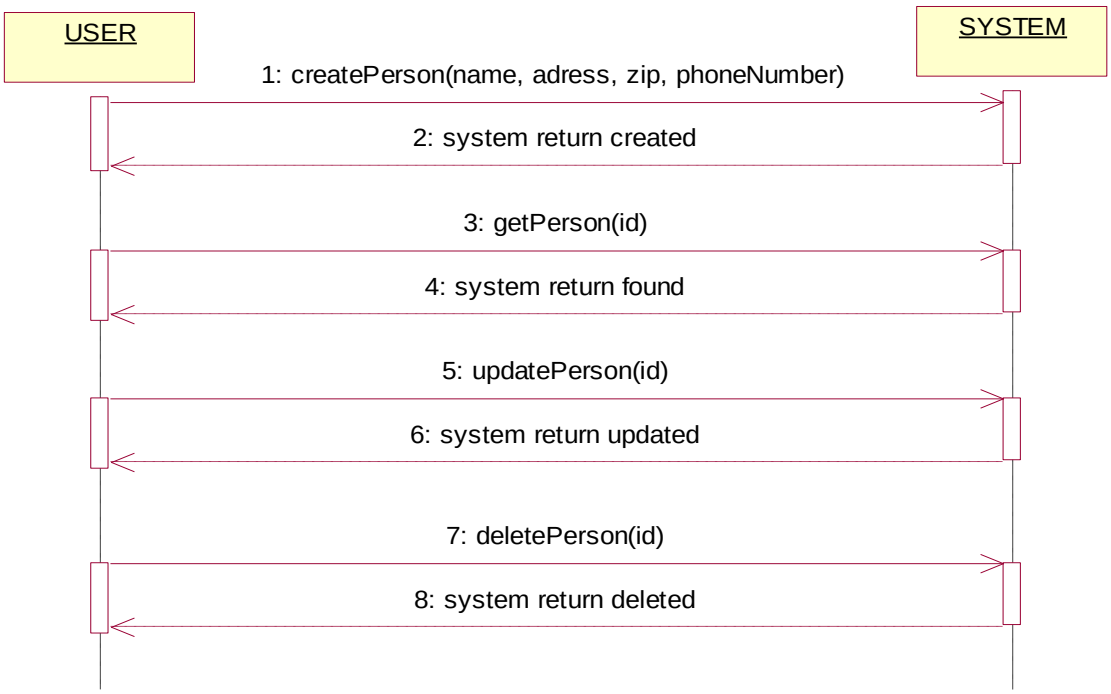
SSD syntaks

Når der er tale om design, bliver systemhændelserne til de metoder, som aktøren bruger i forhold til systemet, derfor angives der også parametre for både inputtet til- og responsen fra systemet – hvis der er nogle parametre.

Operationskontrakter

En operationskontrakt beskriver i detaljer hvilken tilstand systemets objekter efterlades i, efter systemoperationen er udført, i bund og grund kobles use cases (funktionaliteten) og domænemodel en (informationen) her sammen.

Der skrives kontrakter for de operationer der ændrer tilstanden i systemets domæne ved deres udførsel, altså hver gang der er tale om en opdatering.



Ovenstående SSD har følgende operationskontrakter.

Operation: createPerson(name, adresse, zip, phoneNumber)
Use case: Handle Person - CRUD
Type: Update
Pre-betingelser: AdressBook a er oprettet
Post-betingelser:
- personobjektet p blev oprettet
- p blev associeret til a
- p.name, p.adress, p.zip, p.phoneNumber er blevet tilsrkevet værdier

Operation: updatePerson(id)
Usecase: Handle Person - CRUD
Type: Update
Pre-betingelser: personobjekt oprettet og fundet i AdressBook a
Post-betingelser:
- personobjekt p er blevet opdateret og gemt i AdressBook a

Operation: deletePerson(id)
Usecase: Handle Person - CRUD
Type: Update
Pre-betingelser: personobjekt oprettet og fundet i AdressBook a
Post-betingerlser:
- personobjekt p er slettet fra AdressBook a

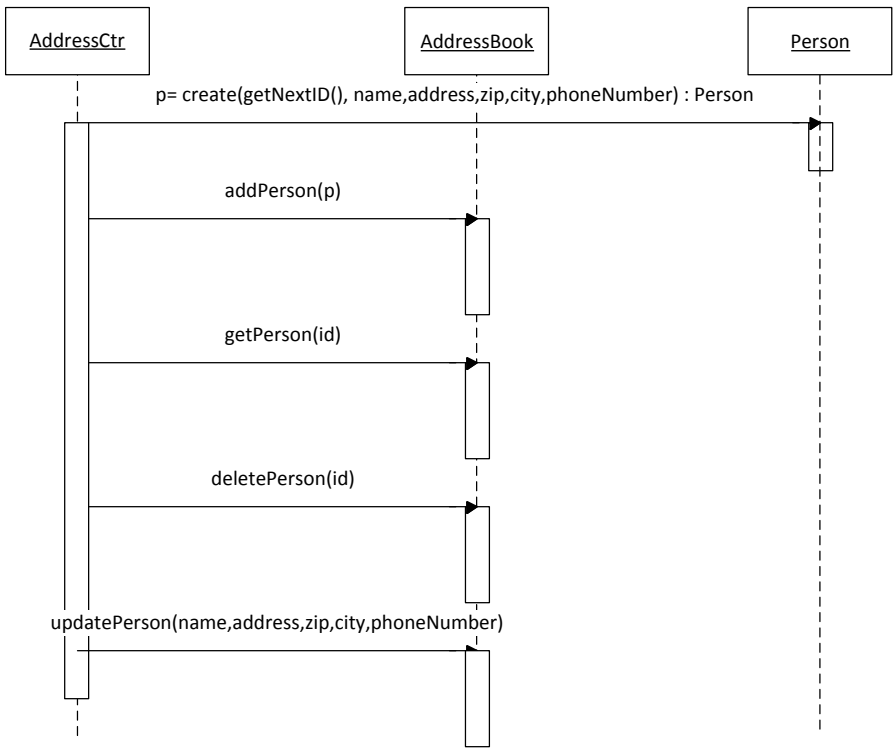
3.3 Interaktionsdiagram

På baggrund af vores SSD, hvor vi finder ud af hvilke metoder, som skal bruges i programmet ville interaktionsdiagrammet vise metoder som er angivet i SSD. Det ville være med til at giv et overblik over, hvilke klasse, som skal kommunikere med hinanden for at create en person. Dette overblik ville gøre det nemmere at programmere, for så ved man hvilke metoder som skal hører til hver klasse.

Sekvensdiagram viser, hvordan vores controller kommunikere med modellaget, hvor CRUD bliver vist frem, Metoden create som først bliver afgivet i AddressCtr med variablene (getNextId(), name, address, zip, city, phoneNumber) som så kalder klassen Person, hvor variablene bliver indsat i Person objektet som så sender det objekt tilbage til AddressCtr, som så tilføjere den til vores samling af personer altså vores add Metode som har objekt Person med.

Metoden Read gør at AddressCtr spørg på et id som en person objekt har, som ligger i vores AdressBook, f.eks. det objekt som vi lavede før, som bliver søgt af AddressCtr. Det objekt som så bliver fundet bliver returneret til AddressCtr objekt.

Update gør at AddressCtr igen spørg på et id som en person objekt har, objekt bliver sendt tilbage til AddressCtr som så redigere (name, address, zip, city, phoneNumber) om person objekt. Metoden Delete skal bruges til at fjerne et person objekt, dette sker ved at AddressCtr søger på et id som person objekt har i AddressBook, når det bliver fundet bliver det så



slettet.

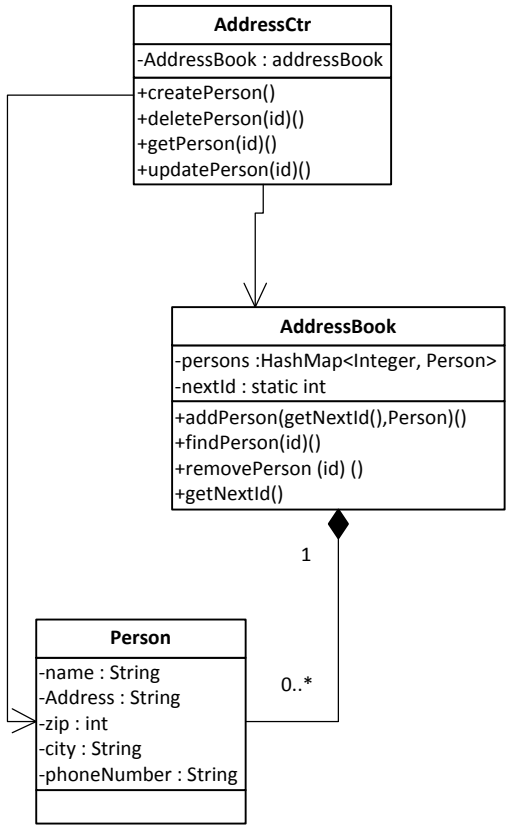
3.4 Designklassediagrammet

På bagrund af Interaktionsdiagrammet skal der udarbejdes et designklassediagram som viser synlighed som hvilke klasser der kommunikere med hinanden, attributter er det som står lige under klassenavnet, datatyper er det som står efter attribut navnet og metoder fra SSD står under attributterne. Dette diagram giver et godt overblik for, hvordan man skal programmer det, her kan man se synlighed for hvilke klasse som kommunikere sammen, hvilke attributter som skal være i den enkle klasse og hvilke metoder, som skal være i den enkle klasse. Dette gør arbejdet med at programmere det en smule nemmere.

I dette designklassediagram bliver det synlig gjort at vi har to ting med fra GRASP

- Ekspert mønstret, som tildelt ansvaret til det objekt der har informationen til at fuldføre det.
- HighCohesion mønstret, som betyder lav binding i metoderne og i klasseren.
- Lav kobling, da vores TUIlager ikke kender til Modellaget, og vores klasser hedder noget sigende, og kun kender til vores public metoder.

Vi kan se i dette designklassediagram at AddressCtr har synlighed til begge klasse, det er for at løse opgaven create person objekt. I diagrammet kan der også se hvilke attributter som der er valgt til hver klasse, og der kan se hvilek metoder som skal bruges for at få programmet til at løse CRUD problemet.



5. Iteration 2

I følgende kapitel beskrives de to use cases ”Handle DVD - CRUD” og ”Handle Copy – CRUD” , ud fra disse skrives SSD’er med systemhændelser og tilhørende operationskontrakter. Ud fra use casen og SSD’en udarbejdes interaktionsdagram, hvorefter vores design klasse diagram opdateres.

5.1 Use case

Handle DVD - CRUD

createDvd ”C”
Primær aktør: User
Pre-betingelse: ingen
Post-betingelse: Et dvd objekt er oprettet
Hoves succes scenarie:

- Brugeren angiver DVD oplysninger(id, titel, director, releaseDate)
- Systemet accepterer oplysningerne og opretter en ny dvd.

findDvd ”R”
Primær aktør: User
Pre-betingelse: Dvd er oprettet
Post-betingelse: Dvd er fundet
Hoves succes scenarie:

- Brugeren angiver det rigtige id
- System accepterer oplysningerne og finder Dvd.
- System returner det rigtig svar.

updateDVD ”U”
Primær aktør: User
Pre-betingelse: Dvd er fundet
Post-betingelse: Systemet opdater Dvd
Hoves succes scenarie:

- Brugeren finder dvd, ændre op oplysninger.
- Systemet accepterer oplysningerne og opdater dvd.

deleteDvd ”D”
Primær aktør: User
Pre-betingelse: Dvd er fundet i systemet og eventuelle eksemplarerne er ikke udlånt
Post-betingelse: Systemet har slettet dvd og eventuelle eksemplarer
Hoves succes scenarie:

- Brugeren finder dvd, og sletter dvd.
- System accepterer oplysningerne og sletter dvd og eventuelle eksemplarer

Handle Copy - CRUD
createCopy"C"
Primær aktør: User
Pre-betingelse: ingen
Post-betingelse: copy er oprettet
Hoves succes scenarie:

- Brugeren angiver copy oplysninger(serialNo, buyDate, buyPrice)
- Systemet accepterer oplysningerne og opretter et ny copy.

findCopy "R"
Primær aktør: User
Pre-betingelse: copy er oprettet
Post-betingelse: copy fundet
Hoves succes scenarie:

- Brugeren angiver det rigtige serialNo
- System accepterer oplysningerne og finder copy.
- System returner det rigtig svar.

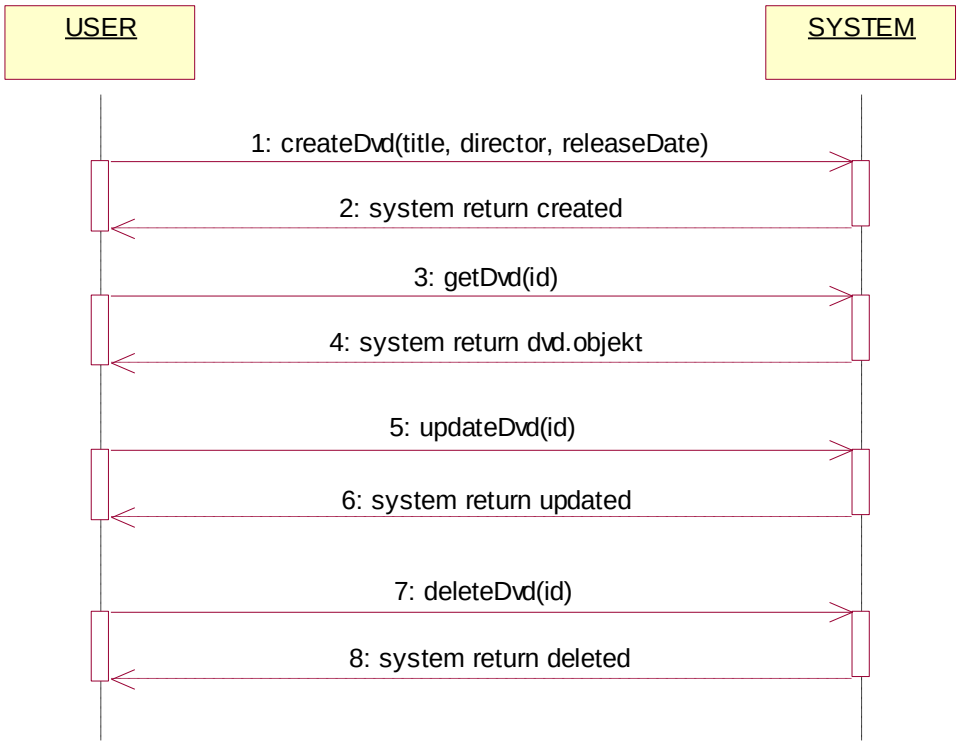
updateCopy "U"
Primær aktør: User
Pre-betingelse: copy er fundet
Post-betingelse: Systemet opdater copy
Hoves succes scenarie:

- Brugeren finder copy, ændre op oplysninger.
- Systemet accepterer oplysningerne og opdaterer copy.

deleteCopy "D"
Primær aktør: User
Pre-betingelse: copy er fundet i systemet.
Post-betingelse: Systemet har slettet copy
Hoves succes scenarie:

- Brugeren finder copy ved hjælp af serialNo, og sletter copy.
- System accepterer oplysningerne og sletter copy

Ovenstående use cases er skrevet som CRUD, og giver anledning til følgende SSD'er samt operationskontrakter for Handle DVD – CRUD og Handle Copy – CRUD.

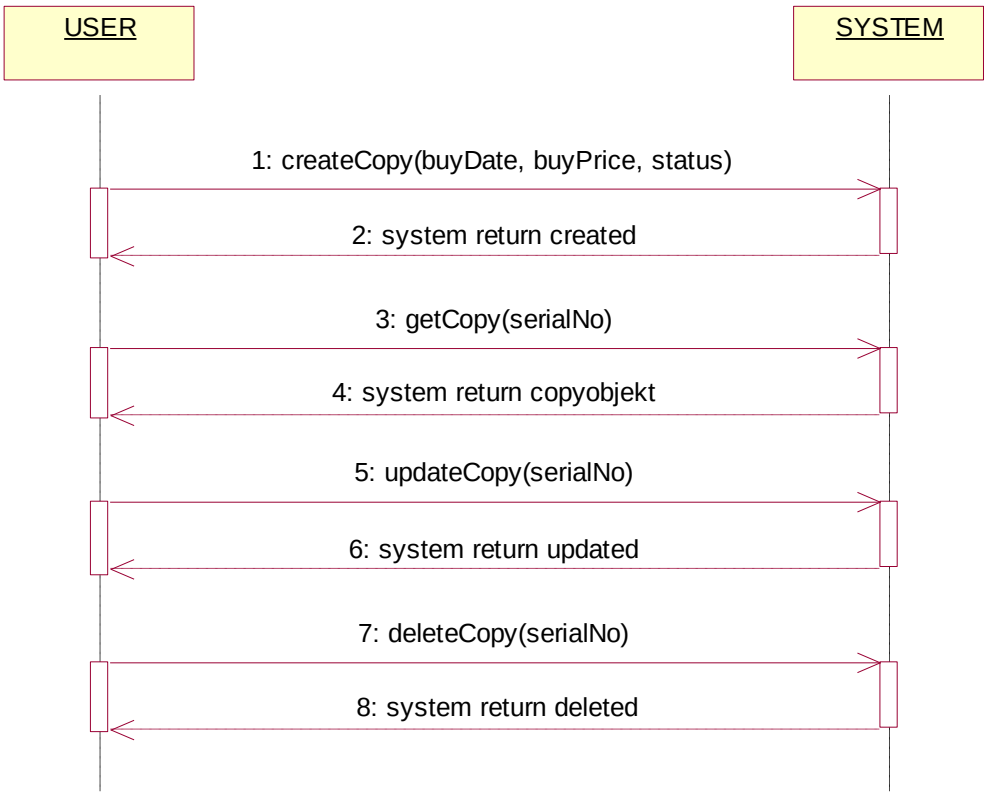


Ovenstående SSD har følgende operationskontrakter.

Operation: createDvd(title, director, releaseDate)
Use case: Handle DVD - CRUD
Type: Update
Pre-betingelser: DVDCollection c er oprettet
Post-betingelser:
- et dvdobjekt dvd blev oprettet
- dvd blev associeret til c
- dvd.title, dvd.director, dvd.releaseDate blev tilsrkevet værdier

Operation: updateDvd(id)
Usecase: Handle DVD - CRUD
Type: Update
Pre-betingelser: DVD oprettet og fundet i DVDCollection
Post-betingelser:
- dvdobjektet er blevet opdateret og gemt i DVDCollection

Operation: deleteDvd(id)
Usecase: Handle DVD - CRUD
Type: Update
Pre-betingelser: DVD oprettet og fundet i DVDCollection
Post-betingerlser:
-dvd.objektet er slettet fra DVDCollection



Ovenstående SSD har følgende operationskontrakter.

Operation: createCopy(buyDate, buyPrice, status)
Use case: Handle Copy - CRUD
Type: Update
Pre-betingelser: Collection c er oprettet
Post-betingelser:
- et copyobjekt copy blev oprettet
- copy blev associeret til c
- copy.buyDate, copy.buyPrice copy.status er blevet tilskrevet værdier

Operation: updateCopy(serialNo)
Use case: Handle Copy - CRUD
Type: Update
Pre-betingelser: Copyobjekt er oprettet og fundet i Collection c
Post-betingelser:
- copyobjektet copy er blevet opdateret og gemt i Collection c

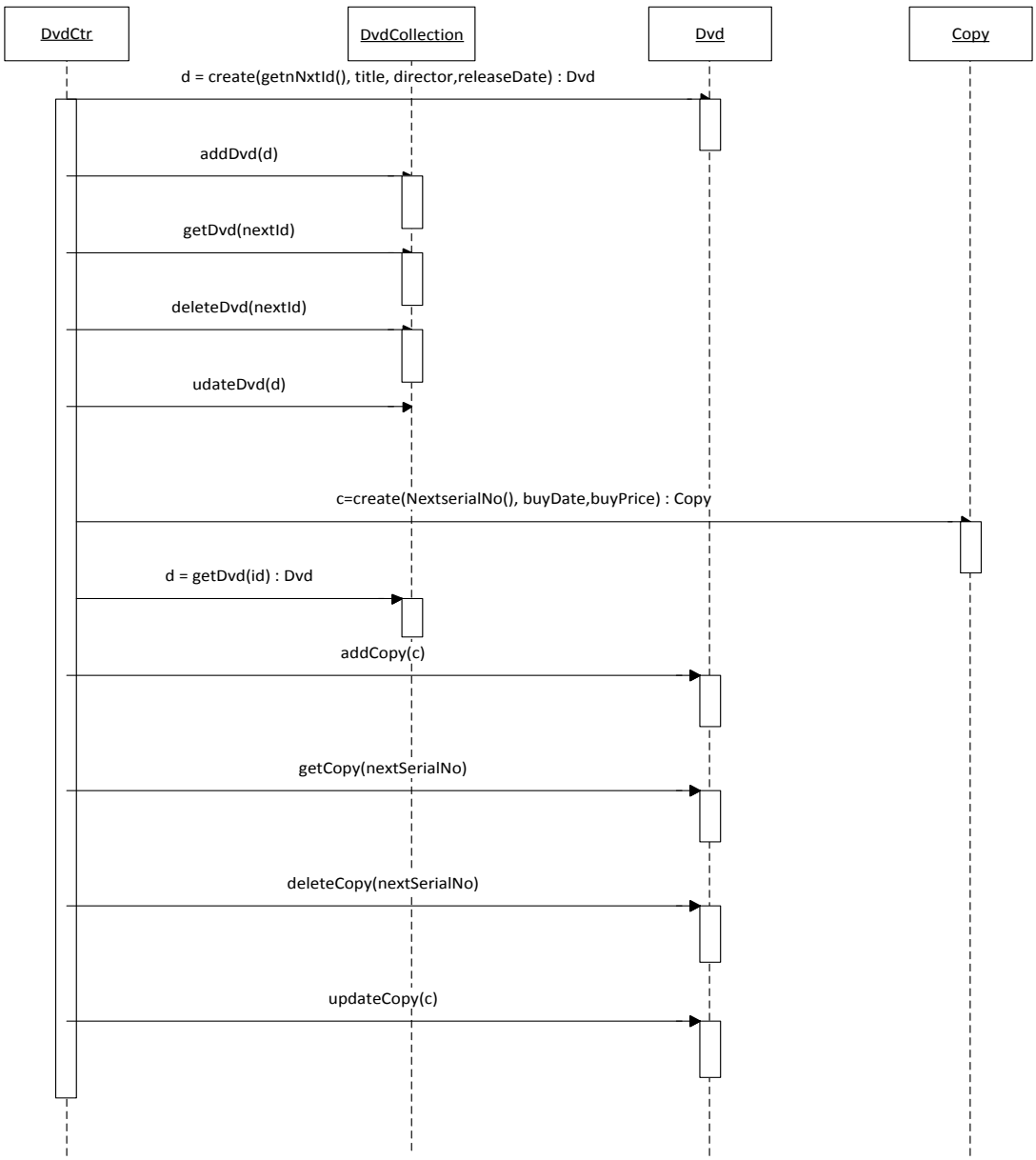
Operation: deleteCopy(serialNo)
Use case: Handle Copy - CRUD
Type: Update
Pre-betingelser: Copyobjekt er oprettet og fundet i Collection c
Post-betingelser:
- copyobjektet copy er blevet slettet fra Collection c

5.2 Interaktionsdiagrammer

Som det først interaktionsdiagram skal dette nye interaktionsdiagram vise, hvordan de metoder som er beskrevet i SSD kommunikere med de forskellige klasser. Det skal være med til at giv et overblik, for at vi skal løse designklassediagrammet. Dette interaktionsdiagram er større fordi der er to CRUD med i en. Det skyldes at Dvd skal have forskellig Copy's ligesom at DvdCollection skal havde forskellig Dvd'er. Så det har givet et bedre overblik at samle det i en.

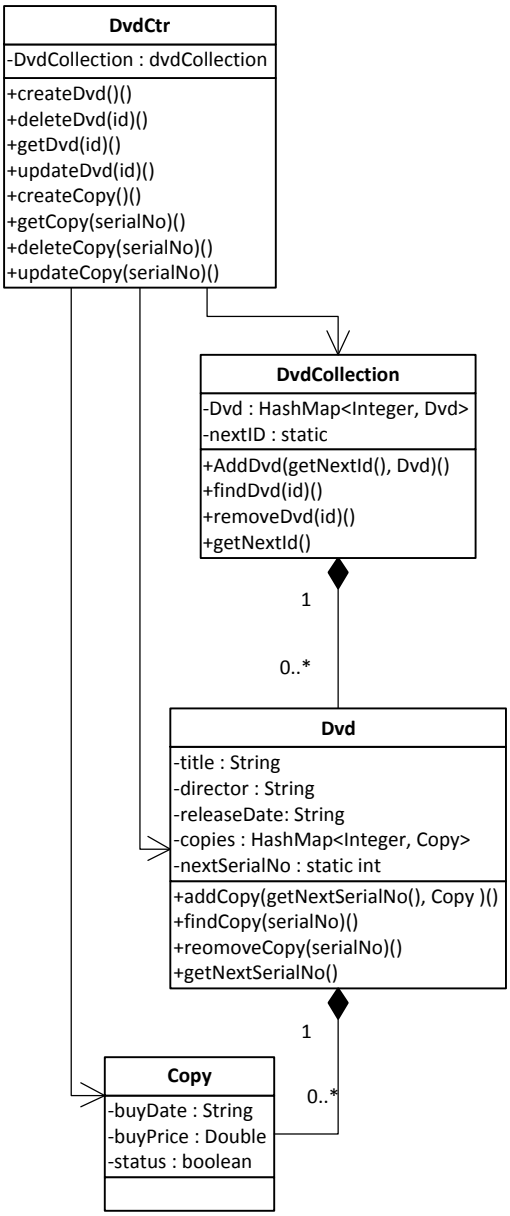
Sekvensdiagram viser næste det samme som i den først bare det er over Dvd i stedet for. Der er de samme metoder CreateDvd, addDvd , getDvd , deleteDvd og updateDvd. Det foregå ligesom med Person objekt. Bare det er over et Dvd Objekt.

Det nye i dette sekvensdiagram er at klasse Copy også er med, det nye i create er, når man holder et objekt af Copy, skal man bare først finde den dvd, som man ville have copy lagt til dvd. Så der bruger vi en metode som er lavt før der hedder getDvd(id), fordi hver dvd har et forskelligt id. Ligesom copy for et forskellig serialNo. Når det dvd objekt er fundet bliver det add en copy til Dvd objekt. Ellers fungere de andre metoder getCopy, deleteCopy og updateCopy på samme måde som de ander bare at man søger i en bestem dvd som man har fra DvdCollection.


DvdCtr
DvdCollection
Dvd
Copy
d = create(getnNxtId(), title, director,releaseDate) : Dvd
addDvd(d)
getDvd(nextId)
deleteDvd(nextId)
udateDvd(d)
c=create(NextserialNo(), buyDate,buyPrice) : Copy
d = getDvd(id) : Dvd
addCopy(c)
getCopy(nextSerialNo)
deleteCopy(nextSerialNo)
updateCopy(c)

5.3 Designklassediagrammet.

Ligesom det først Designklassediagram skal vi bruge det fra interaktionsdiagrammet, som har givet et overblik over hvilke klasse, som skal bruge hvilke metoder. Forskellen på dette designklassediagram er at klassen dvd indeholder forskellig copy's men det er stadig DvdCtr som styre det hele, og har synlighed til det hele. Der er stadig Ekspert mønstret, HighCohesion og lav kobling som bliver brugt fra GRASP. Ellers er der ikke den stor forskelle på de to diagramme ud over navn på klassen, attributter og metoder.



6. Iteration 3

I følgende kapitel beskrives de to use cases ”Loan DVD” og ”Return DVD” fra Iteration 4, da vi så det som en naturlig proces, at integrere Loan og Return DVD i hinanden. Ud fra disse use cases skrives SSD’er med systemhændelser og tilhørende operationskontrakter, og herefter udarbejdes interaktionsdagrammer, hvorefter vores design klasse diagram opdateres.

6.1 Use case

De to use cases ”Loan DVD” og ”Return DVD” har vi valgt at skrive fully dressed – dette på funktionerne createLoan() og returnLoan(), da disse to metoder er de mest komplekse.

Use case – Loan DVD

Use-case: createLoan() id : UC4

Aktør : User

Pre betingelser: En Person, en DVD med tilhørende eksemplar er oprettet i systemet.

Post betingelser: Udlånet er registreret i systemet.

Frekvens: -

Basis succes flow

1. En person henvender sig for at låne en dvd.
2. Personens ID indtastes i systemet.
3. Systemet finder personen.
4. Der angives hvilken DVD der ønskes lånt.
5. Systemet melder tilbage at der er ledige eksemplarer.
6. Der angives ønsket låne-periode.
7. Systemet registrerer perioden, og opretter det endelige lån.

Alternative scenarier

1. Til enhver tid hvor systemet melder fejl:
   Til at støtte korrekt administrering af udlån, skal systemet kunne gendanne det arbejde man lige har siddet med.
   a. Brugeren genstarter systemet → Systemet spørg om tidligere arbejde skal gendannes.
   b. Brugeren svarer Ja/Nej og systemet reagerer herefter.
2. Personen findes ikke i systemet:
   a. Systemet gør opmærksom på, at personen ikke findes.
   b. Brugeren opretter herefter personen.

3. DVD'en findes ikke i systemet
    a. Systemet gør opmærksom på, at DVD'en ikke findes.
4. DVD'en eksisterer, men der er ingen eksemplarer hjemme.
    a. Systemet gør opmærksom på, at der ikke er nogle eksemplarer til rådighed.

Use case – Return DVD

Use-case: returnLoan() id : UC5

Aktør : User

Pre betingelser: Et udlån er oprettet i systemet.

Post betingelser: Afleveringen er registreret i systemet.

Frekvens: -

Basis succes flow

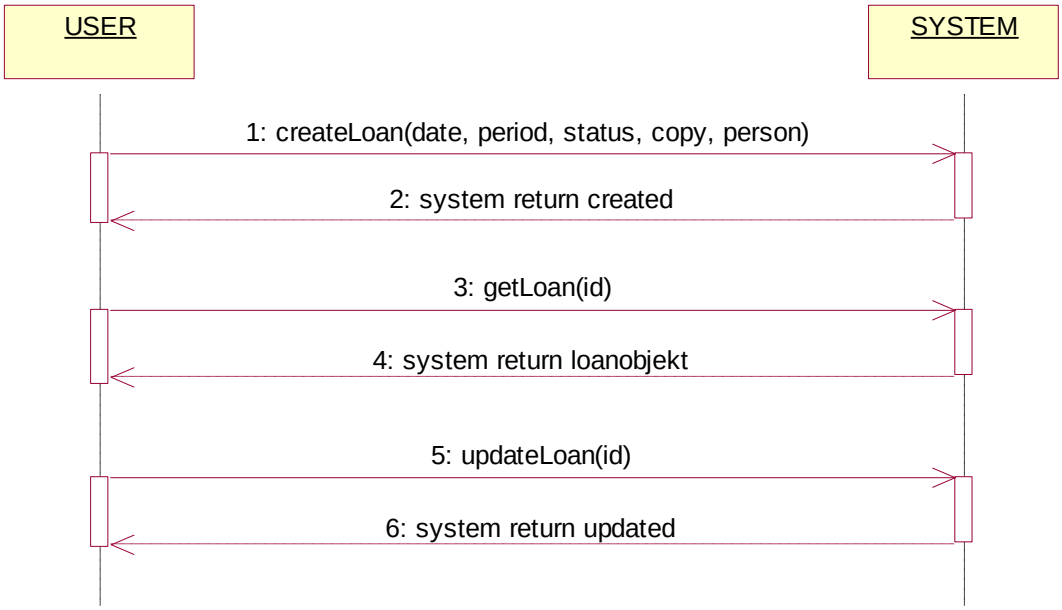
8. En person henvender sig for at aflevere et DVD-eksemplar.
9. Serie-nummeret på DVD-eksemplaret indtastes.
10. Systemet finder lånet.
11. Lånet skifter status fra at være aktivt til at være afsluttet.
12. DVD-eksemplaret skifter status fra udlånt til frit.
13. Afleveringen registreres, og en kvittering udskrives.

Alternative scenarier

5. Til enhver tid hvor systemet melder fejl:
   Til at støtte korrekt administrering af aflevering, skal systemet kunne gendanne det arbejde man lige har siddet med.
    a. Brugeren genstarter systemet → Systemet spørg om tidligere arbejde skal gendannes.
    b. Brugeren svarer Ja/Nej og systemet reagerer herefter.
6. Det indtastede serie-nummer på DVD-eksemplaret findes ikke i systemet.
    a. Systemet gør opmærksom på, at serie-nummeret ikke eksisterer.
    b. Brugeren slår personen op på personens låne-id, og sletter lånet manuelt.

Ovenstående use cases, skrevet fully dressed, giver anledning til følgende SSD'er samt operationskontrakter for Loan DVD og Return DVD.

SSD – Loan DVD



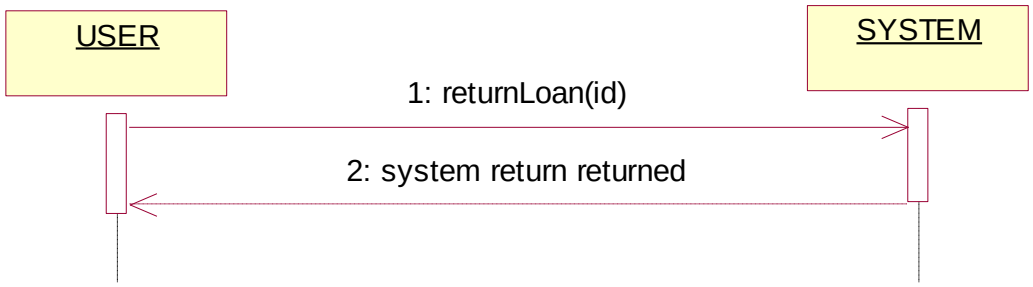
Ovenstående SSD har følgende operationskontrakter.

Operations kontrakter:

Operation: createLoan(date, period, status, copy, person)
Use case: Handle Loan
Type: Update
Pre-betingelser: Collection c er oprettet
Post-betingelser:
- et loanobjekt lend blev oprettet
- loan blev associeret til c
- loan.date, loan.period, loan.status loan.copy, loan.person er blevet tilskrevet værdier

Operation: updateLoan(id)
Use case: Handle Loan
Type: Update
Pre-betingelser: Loanobjekt er oprettet og fundet i Collection c
Post-betingelser:
- loanobjektet loan er blevet opdateret og gemt i Collection c

SSD – Return DVD



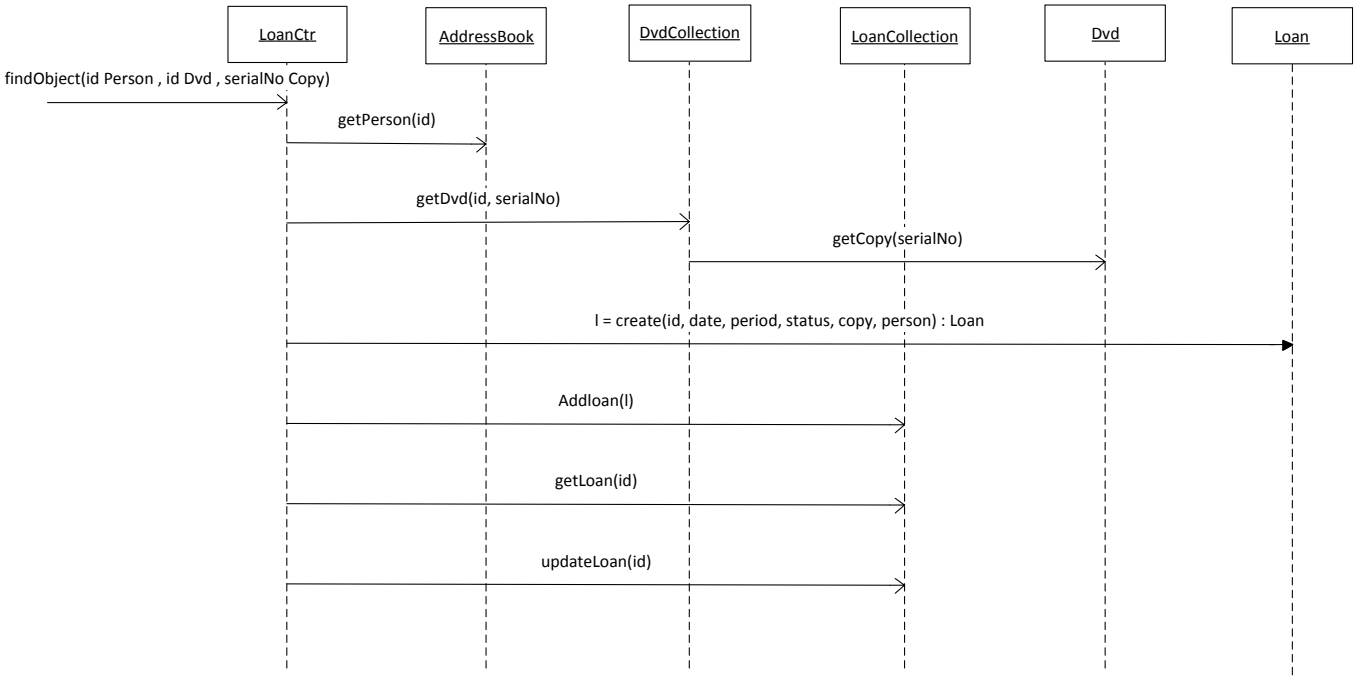
Ovenstående SSD har følgende operationskontrakter.

Operation: returnLoan(id)
Usecase: Return DVD
Type: Update
Pre-betingelser: At DVD'en er udlånt og lånet er oprettet
Post-betingerlser: Lånet er afsluttet og DVD eksemplaret er til rådighed i systemet
- loan.setStatus ændres fra true til false
- loan.getCopy.setStatus ændres fra false til true

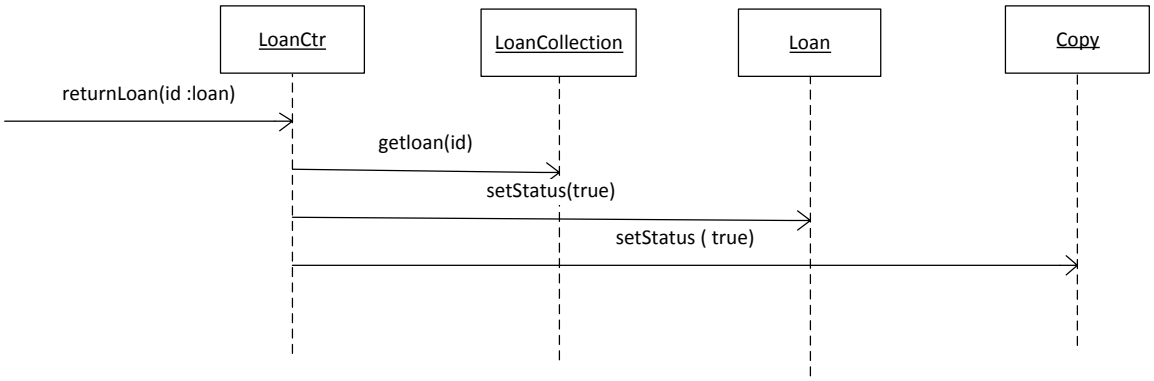
6.2 Interaktionsdiagrammer

I følgende sekvensdiagram er det ikke en CRUD mere det er ikke en simple funktion mere, den har andre klasser med. Men stadig henter vi metoderen fra SSD og så angiver vi her hvilke klasser som skal snakke sammen for at løse metoden. Dette giver et bedre overblik over, hvordan systemet skal kommuniker. Sekvensdiagram viser, hvordan vores controller kommunikere med modellaget,

create et Loan er vi nød til først at finde en Person og en Copy. Det er her at det ikke er simple funktion mere. Når LoanCtr har de to objekter kan man create et Loan med de nødvendige oplysninger.
Read at man søger på det id som Loan har.
Update er også at søge på id og så redigere de nye oplysninger

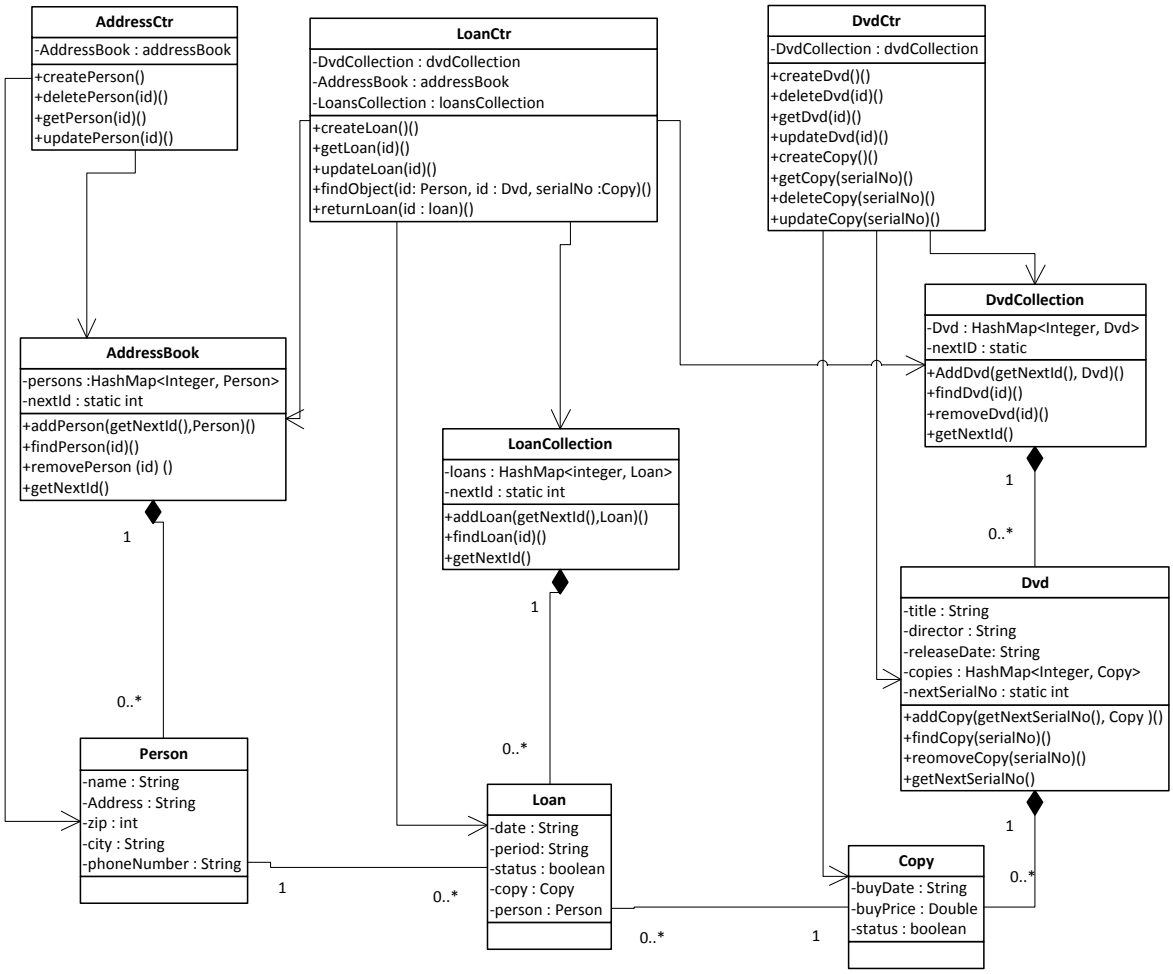


I understående sekvensdiagram bliver der vist hvordan systemet skal håndtere returnLoan, da man har tænkt på at man skulle returnLoan, har man tilføjet de rigtig klasser så vi kan godt return uden at lave en ny klasse, da vi kun skal tilføje en ny metode i vores LoanCtr.



6.3 Designklassediagram

Som sagt i de andre designklassediagrammer giver diagrammet et bedre overblik over metoder, datatyper og synligheden, viser også hvordan GRASP bliver brugt. Det som man kan se ud fra diagrammet hvor GRASP bliver synlig gjort er at der lagt vægt på Ekspert mønstret, som tildelt ansvaret til det objekt der har informationen til at fuldføre det. High Cohesion mønstret som betyder lav binding i metoderne og i klasseren. Vi har også lav kobling da TUIlaget kun kender til controlleren, og controlleren kun kender modellaget, og klasseren kun kender til public metoder.
Designklassediagrammet for alle klasser her er angivet de nødvendige attributter og metoder til de forskellige klasser. Vores nye controller LoanCtr har synlighed til DvdCollection, LoanCollection, AddressBook og Loan, for at samle metoderen i controllerlaget, da det giver et bedre overblik. Her kan også ses at de to tidligere diagrammer er sat sammen ved hjælp af Loan klassen og LoanCtr klassen.


AddressCtr
-AddressBook : addressBook
+createPerson()
+deletePerson(id)()
+getPerson(id)()
+updatePerson(id)()
LoanCtr
-DvdCollection : dvdCollection
-AddressBook : addressBook
-LoansCollection : loansCollection
+createLoan()()
+getLoan(id)()
+updateLoan(id)()
+findObject(id: Person, id : Dvd, serialNo :Copy)()
+returnLoan(id : loan)()
DvdCtr
-DvdCollection : dvdCollection
+createDvd()()
+deleteDvd(id)()
+getDvd(id)()
+updateDvd(id)()
+createCopy()()
+getCopy(serialNo)()
+deleteCopy(serialNo)()
+updateCopy(serialNo)()
DvdCollection
-Dvd : HashMap<Integer, Dvd>
-nextID : static
+AddDvd(getNextId(), Dvd)()
+findDvd(id)()
+removeDvd(id)()
+getNextId()
AddressBook
-persons :HashMap<Integer, Person>
-nextId : static int
+addPerson(getNextId(),Person)()
+findPerson(id)()
+removePerson (id) ()
+getNextId()
LoanCollection
-loans : HashMap<integer, Loan>
-nextId : static int
+addLoan(getNextId(),Loan)()
+findLoan(id)()
+getNextId()
Dvd
-title : String
-director : String
-releaseDate: String
-copies : HashMap<Integer, Copy>
-nextSerialNo : static int
+addCopy(getNextSerialNo(), Copy)()
+findCopy(serialNo)()
+reomoveCopy(serialNo)()
+getNextSerialNo()
Person
-name : String
-Address : String
-zip : int
-city : String
-phoneNumber : String
Loan
-date : String
-period: String
-status : boolean
-copy : Copy
-person : Person
Copy
-buyDate : String
-buyPrice : Double
-status : boolean
1
0..*
1
0..*
1
0..*
1
0..*
1
0..*
1
0..*
1
0..*
1

7. Implementering

I det følgende kapitel bliver der givet forklaringer på nogle af de beslutninger og udfordringer, som har fundet sted i selve programmeringsdelen af dette projekt. Først beskrives processen, som har fundet sted i forbindelse med at komme fra design til implementering. Efterfølgende bliver der givet en forklaring på brug af Singleton princippet, mere præcist hvorfor og hvordan det virker i forbindelse med dette projekt. Derefter bliver der givet en beskrivelse af nogle af de udfordringer der er opstået undervejs, samt hvilke beslutninger de har medført og indflydelsen de har haft på udformningen af programmet. Til sidst bliver der udført en detaljeret unittest.

7.1 Design til implementering

Den generelle retningslinje for fremgangsmåden har været at, oversætte direkte fra vores designklassediagram til selve kodedelen. Dog skal det her nævnes at selve koden, og udformningen af denne, hele tiden har været en medspiller i hvordan designklassediagrammet er kommet til at se ud. Dette er noget som har fundet sted automatisk, og har både været en fordel og en ulempe. Fordelen ligger i at der ikke er blevet designet noget, som gruppen har været ude af stand til at programmere, mens ulempen er at udformningen af designet muligvis har været begrænset netop af samme grund.

7.2 Singleton

Formålet med Singleton princippet er at sikre at der kun kan instantieres et objekt af en given klasse. Dette kan med fordel benyttes i forbindelse med personkartoteker, eller samlinger af den ene eller anden art.

Dette princip bliver benyttet både i modellaget og i controllerlaget i projektet, og sikre derfor at der kun findes et objekt af klasserne AddressBook og DvdCollection, samt et objekt af hver controllerklasse. Princippet sørger derfor for at når der er flere controllerklasser som har adressebogen eller dvd-samligen med som instansvariabel, så er det den samme adressebog begge steder, og ikke to forskellige instanser.

7.3 Udfordringer

Som nævnt i 7.1 så har gruppen forsøgt at udarbejde designet med selve programmeringen i baghovedet. Dog er der steder hvor designbeslutningerne ikke har været velovervejet og derfor har resulteret i senere ændringer af hele koden samt designdiagrammerne. Et eksempel på dette er omkring ArrayList kontra HashMap. Indledningsvis var både adressebogen, dvd-samlingen samt kopier af de forskellige dvd'er gemt i ArrayLists, men denne beslutning virkede ikke hensigtsmæssig da alle førnævnte objekter indeholder et unikt ID. Derfor besluttede vi at ændre formen til HashMap, så dette unikke ID fungerer som nøgle til værdien, som så er objektet. Resultatet af denne beslutningen var ændringer igennem mange metoder, og klasser som ellers fungerede, samt ændringer igennem mange diagrammer i designdelen.

7.4 Unittest

Tests er en stor del af selve implementeringen, og i dette afsnit bliver der gået i dybden med unittest på at oprette en lån i programmet. Denne form for unittesting vil, under normale omstændigheder, blive udarbejdet for samtlige klasser og metoder, men dette har ikke været muligt på grund af den begrænsede projektperiode. Unittesten er udarbejdet på baggrund af use casen createLoan, og vil i sidste ende resultere i at der er taget højde for samtlige input i forbindelse med denne operation.

7.4.1 Beskrivelse af test

Som nævnt ovenfor, så bliver use casen createLoan benyttet her for at finde frem til de metoder som kræver input i forbindelse med oprettelsen af et lån. Disse metoder skal derefter testes med alle tænkelige inputs, heriblandt grænseværdier, positive- og negative tal, referencer til objekter der findes og ikke findes og så videre. Metoderne og de input som skal testes er indsat i skemaet nedenfor.

Loan DVD	Input	Forventet output
2. Personens ID indtastes i systemet	-1	Fejl, findes ikke negativ ID
-	1	Objekt af Person ID 1: John Hans
-	50	Fejl, findes ingen objekt med ID 50
-	h	Skriver ”Indtast et nummer”
4. Der angives hvilken DVD der ønskes lånt	-1	Melder fejl, lån bliver ikke orpettet
-	1	Objekt af Dvd ID 1: Die hard
-	50	Melder fejl, lån bliver ikke orpettet
-	h	Fejl, forventer en int
x. Angiv hvilken copy der ønskes	-1	Fejl, findes ikke negativ serialNo
-	1	Objekt af Copy serialNo 1
-	50	Fejl, findes ingen objekt med serialNo 50
-	h	Fejl, forventer en int

Efter udarbejdelsen af ovenstående diagram, testes programmet med de forskellige input ved de forskellige metoder. Det faktiske output noteres ned, og bruges til sammenligning med de forventede output.

Loan DVD	Input	Faktiske output
2. Personens ID indtastes i systemet	-1	Ingen fejl, opretter lån men Person er null
-	1	Objekt af Person 1: John Hans
-	50	Ingen fejl, opretter lån men Person er null
-	h	Skriver ”Indtast et nummer”
4. Der angives hvilken DVD der ønskes lånt	-1	Melder fejl, lån bliver ikke orpettet
-	1	Objekt af Dvd ID 1: Die hard
-	50	Melder fejl, lån bliver ikke orpettet
-	h	Fejl, forventer en int
x. Angiv hvilken copy der ønskes	-1	Ingen fejl, opretter lån men Copy er null
-	1	Objekt af Copy serialNo 1
-	50	Ingen fejl, opretter lån men Copy er null
-	h	Fejl, forventer en int

Efter sammenligningen mellem de forventede output og de faktiske output kan det konkluderes at der er steder hvor de to stemmer overens, mens der er andre steder hvor det faktiske output er helt anderledes end forventningen. Resultaterne af testen benyttes så til at ændre programmet til at kunne overkomme de fejl som testen påpeger. De steder hvor det forventede output og det faktiske output begge ender med fejl, bør koden redigeres til fejlen ikke længere opstår, og de steder hvor fejler kommer som en overraskelse bør fejlen findes og fjernes. Efter diverse ændringer køres endnu en test indtil resultaterne er som ønsket.

Ud fra vores test, er det mest overraskende resultat at der ikke kommer en fejl når der forsøges at låne ud til en person med ID -1 og ID 50 som ikke findes. Her opretter programmet et Loan men hvor person er null. Det samme sker med Copy serialNo -1 og 50. Her skal koden ændres til at lave et tjek på om der rent faktisk eksisterer objekter af typen person og copy, med det indtastede ID og eller serialNo.

8. Gruppe evaluering

Hvordan fungerer gruppearbejdet generelt?
- Generelt fungerede gruppearbejdet godt. Alle har hver deres mening, hvilket har skabt nogle gode debatter, og mundet ud i et resultat alle kan stå inde for. Samtidig medfører debatterne, at opgavens enkle elementer er blevet gennemarbejdet, og alle har haft indflydelse.

Hvordan er den overordnede tilfredshed med gruppearbejdet? Har I det godt sammen?
- Overordnet er vi tilfredse med gruppearbejdet, vi har en god fælles humor, og samtidig ved gruppen også, hvornår der skal arbejdes.

Hvordan er informationsflowet i gruppen?
- Godt, alle har deltaget aktivt, og vi har sørget for at holde hinanden opdateret på hvad vi har nået, og hvad vi skal nå i morgen. Dog kunne tidsplanen have været brugt bedre som rettesnor for rapporten.

Hvordan fungerer samarbejdet i forhold til nedenstående punkter:

- Samarbejde (Collaboration)
    - Hvordan er dialogen i gruppen? Er der åbenhed og dialog og påvirker I hinandens holdninger? Kommer alle frem med deres mening? Er der nogen der er for dominerende? Er der nogen, der overhovedet ikke kommer på banen?
      - Alle deltog aktivt, ingen decideret leder i gruppen, alle medvirkende til en god opgaveløsning, ved at debattere mulige løsninger. Når spørgsmål er opstået i gruppen, har vi taget den på gruppe-niveau samt adspurgt en lærer.
- Samordning (Coordination)
    - Har I diskuteret målsætning for gruppen omkring ambitionsniveau, arbejdsindsats, mm?
      - I forbindelse med fastlæggelsen af gruppens krav til hinanden, diskuterede vi også målsætninger og ambitioner for gruppen – disse kom vi hurtigt til enighed om.
    - Sørger I for at der skabes helhed i samarbejdet og i produktet? (skabes der synergi?)
      - På baggrund af at alle har været aktive i gruppen, og løsningsforslagene er blevet diskuteret, er opgaven blevet løst som en helhed.
- Samhørighed (Coalition)
    - Føler i jer som en gruppe med tilhørsforhold og gruppeånd?
      - Bestemt, god humor kombineret med seriøst arbejde har bidraget til god samhørighed.
- Styring (Control)
    - Hvordan fungerer gruppeledelsen? Er der respekt for de aftaler som træffes i gruppen og de grupperegler I har sat op?
      - Bestemt, aftaler er blevet overholdt, indbyrdes respekt i gruppen.

9. Konklusion

Opgaven for gruppen var, at udvikle en applikation med en adressebog, der indeholder oplysninger om venner, samt oplysninger om en DVD samling, hvorfra det skal være muligt at udlåne og aflevere DVD'er.

For at komme til selve implementeringen af koden, var det en nødvendighed at gennemarbejde og analysere selve opgaven, for at designe de forskellige klasser og metoder – samt på hvilken måde de skulle kommunikere. Dette har resulteret i, at gruppen har haft stor fokus på, at selve implementeringen stemte godt overens med vores endelige design klasse diagram.

Gruppen har forsøgt på at holde arbejdet opdelt i 3 iterationer, for at simplificere arbejdet, samt at gøre det lettere at implementere koden i små bidder. Dog oplevede gruppen i starten af 3. iteration komplikationer med den 3-delte arkitektur (model view controller), da det tog mere tid end planlagt, at få TUI-laget til at kalde de forskellige metoder ned igennem samtlige lag, samt at få vist det ønskede resultat på skærmen.

Alt i alt har det resulteret i, at gruppen har forsøgt på at færdiggøre designdelen, deriblandt også muligheden for at aflevere et lån, men på grund af ovenstående komplikationer, løb gruppen tør for tid – hvilket betyder, at programmet ikke er færdig kodet.