TEMA DESIGN

10. NOVEMBER 2017
Projekt rapport

Jacob Thomsen

Jonas Weile

Daniel Pedersen

Martin Lundgren

Peter Madsen

Professionshøjskolen

University College

Nordjylland

Sofiendalsvej 60

9200 Aalborg SV

Datamatiker

University College Nordjylland

Teknologi og Business

Datamatiker

dmab0917

Tema Design

Antal anslag: x

Antal siger i alt: x

Projektdeltagere:

Jacob Thomsen

Jonas Weile

Daniel Pedersen

Martin Lundgren

Peter Madsen

Vejleder:
Anita Lykke Clemmensen

István Knoll

Afleveringsdato:
10-11-2017

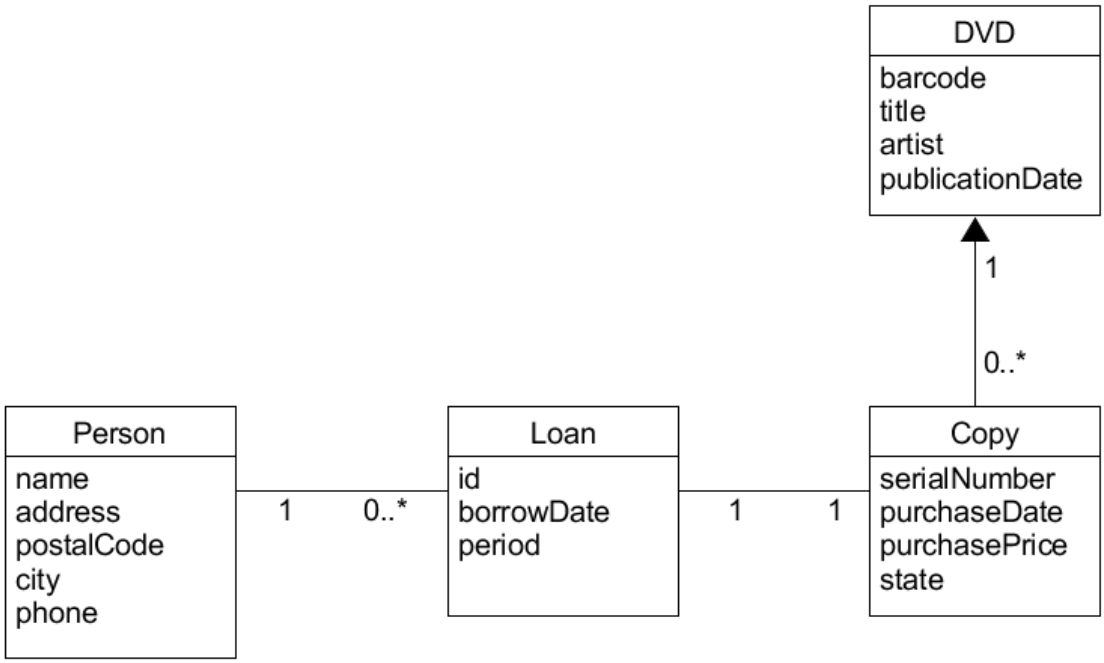
Indholdsfortegnelse

Introduktion

I denne rapport skal der løses Preliminary work, som indebærer en domænemodel, kode standard og projektets arkitektur. Der skal laves 4 iterationer og der er mulighed for at lave tilføjelser efterfølgende, såsom reserver.
Systemet indebærer, at der kan oprettes låner, lave DVD samt kopier af dem og have mulighed for at låne disse kopier ud og at kunne returnere dem.

Preliminary work

Domænemodel



Dette er domænemodellen for systemet. En person kan have 0 til mange lån. Lånet kan kun have en person tilknyttet til sig og hvert lån kan have en kopi. En kopi kan have knyttet et lån til sig og en DVD. En DVD kan have knyttet 0 til mange kopier til sig. Attributten state er blevet flyttet fra loan til copy, da en kopis ledighed er knyttet til en kopi og ikke et lån.

Gruppens kode standard

I dette projekt vil der blive programmeret efter de officielle Java 8 standarder. Afviges der fra disse, vil det blive begrundet. Dette er for at holde koden mest muligt forståelig for andre programmører, her specifikt opponentgruppen.

Projekts arkitektur

Projektets arkitektur er delt op i en 3-lags arkitektur. Et userinterface lag, dette er grænsefladen, et controllerlag der indeholder funktionerne samt et modellag der indeholder egenskaber, herunder containere.

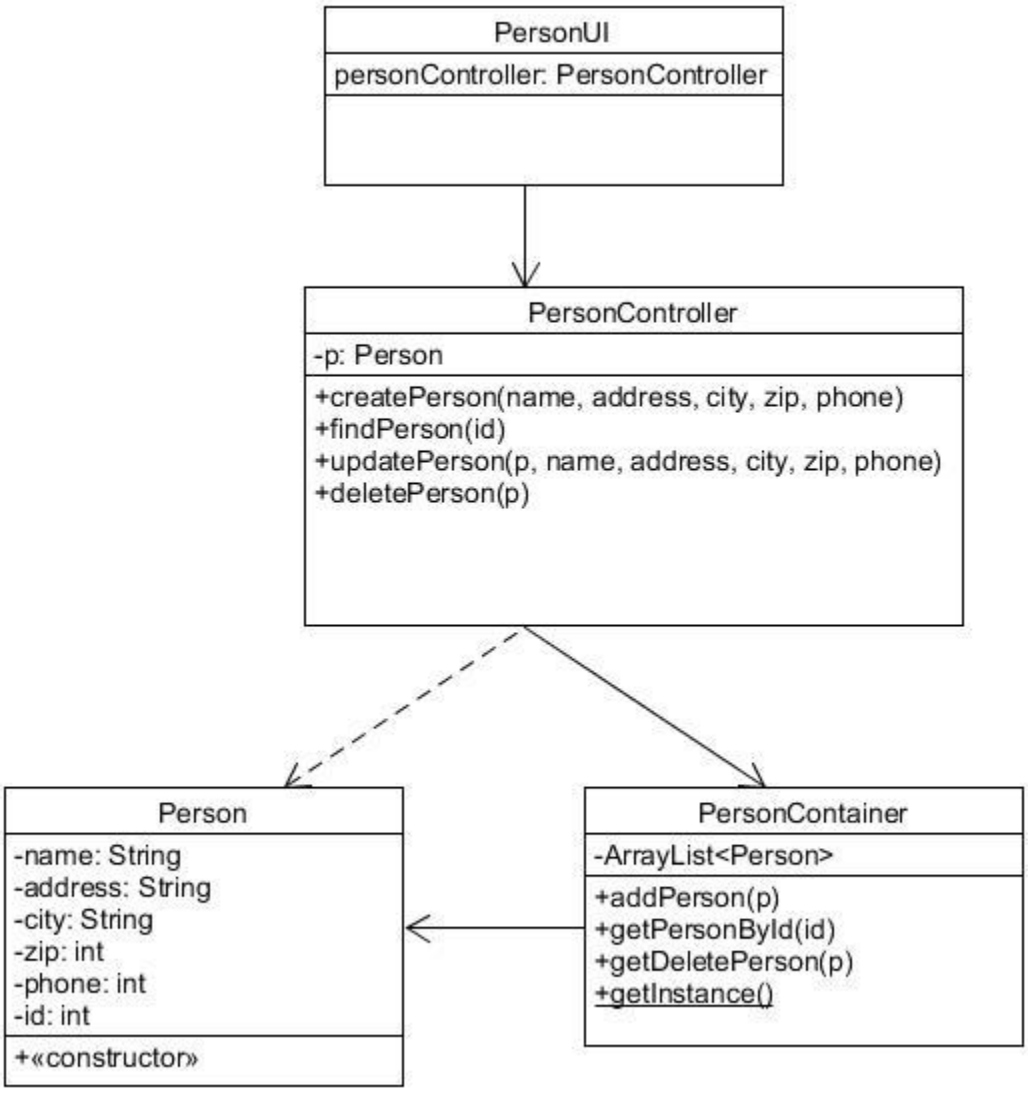
Iteration 1

Introduktion

I første iteration skal det være muligt at lave CRUD på Person.

Diagrammer

Designklasse diagram



Dette er et designklasse diagram ud fra use casen ”Manage Person - CRUD”. Her ses den nævnte 3-lags arkitektur, hvor vi i toppen har UI, som har adgang til Controlleren, hvor controlleren har adgang ned til modellaget som indeholde Container og klassen selv, i dette tilfælde *Person*. På designklasse diagrammet, er der angivet, klasser, metoder, synlighed og datatyper. Dette princip gør sig gældende for de resterende designklasse diagrammer i rapporten.

Delkonklusion

I første iteration blev det muligt at lave CRUD på personer, der i det fulde system skal være lånere. Det er dog ikke den endelige udgave af denne del af programmet. Det ville f.eks. være ønskeligt at kunne se hvilke lån en person har og hvilke DVDer, der hører til disse. Dette vil blive tilføjet i en senere iteration.

Iteration 2

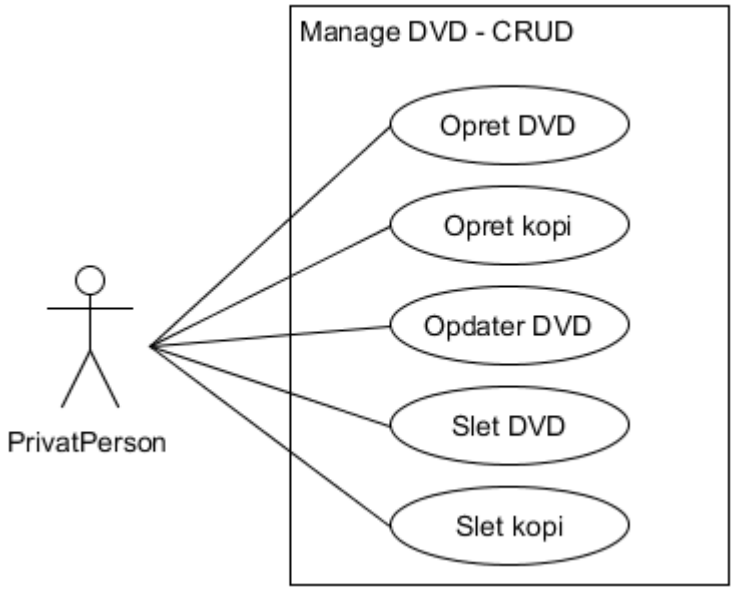
Introduktion

I anden iteration af programmet skal det blive muligt at oprette DVD’er og eksemplarer af dem. DVD klassen skal indeholde de abstrakte informationer som titel og barkode. Kopier er de konkrete DVD’er, som faktisk skal kunne udlånes. De informationer, som kopier skal holde på er forbindelse til en DVD og praktiske informationer, såsom serienummer.

Diagrammer

Use case

Dette er use casen ”Manage DVD - CRUD”.

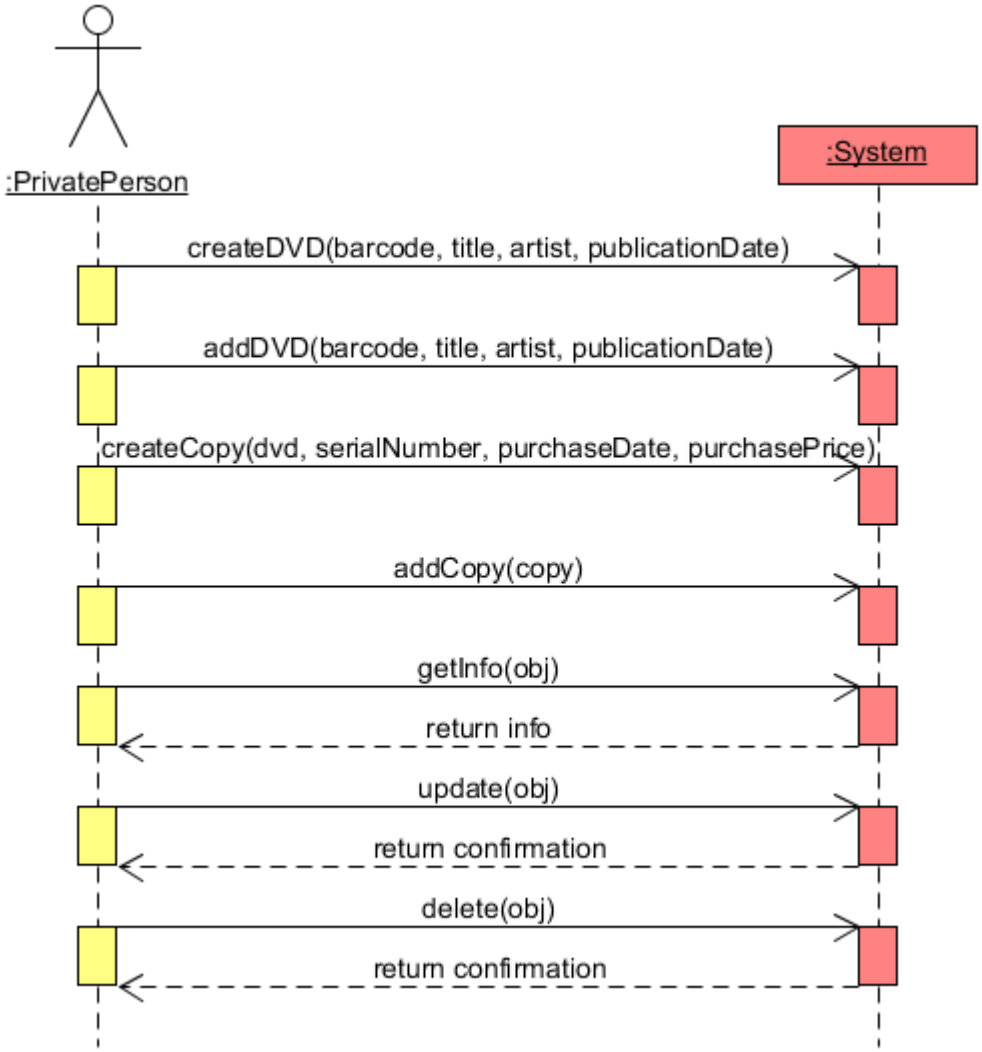


Fully dressed use case

Use case navn	Manage DVD - CRUD	
Aktører	Private Person	
Præbetingelser	Der udkommer en ny DVD	
Postbetingelser	En DVD er blevet oprettet, aflæst, opdateret og slettet	
Frekvens	0-1 gange om dagen	
Flow of events	Aktør	System
	1. PrivatePerson tjekker om der udkommer ny DVD	
	2. PrivatePerson køber en kopi af DVDen og opretter den i systemet	3. System opretter DVDen og angiver at dette er udført
	4. PrivatePerson tilføjer et unikt serienr til DVDen, så aktøren kan holde styr på flere kopier	5. Systemet opretter en kopi
	6. PrivatePerson opdager fejl i oplysningerne om DVDen og vil rette dem	7. Systemet henter oplysningerne og anmoder om nye detaljer
	8. PrivatePerson indskriver nye oplysninger	9. Systemet overskriver de gamle med de nye oplysninger og giver besked at det er udført

	10. PrivatePerson finder ud af at ingen vil låne kopier af DVDen og vil derfor slette den fra systemet	11. Systemet henter oplysninger om DVDen og sletter den og kopierne fra systemet, giver besked om slettelse
Alternative flow	1a. Ingen ny DVD udkommer så ingen bliver oprettet 2a. Kan ikke anskaffe en kopi af DVD'en 4a. PrivatePerson indtaster et serienr, der allerede eksisterer 1. Se punkt 8. 10a. PrivatePerson kan ikke finde DVDen aktøren vil slette	

SSD



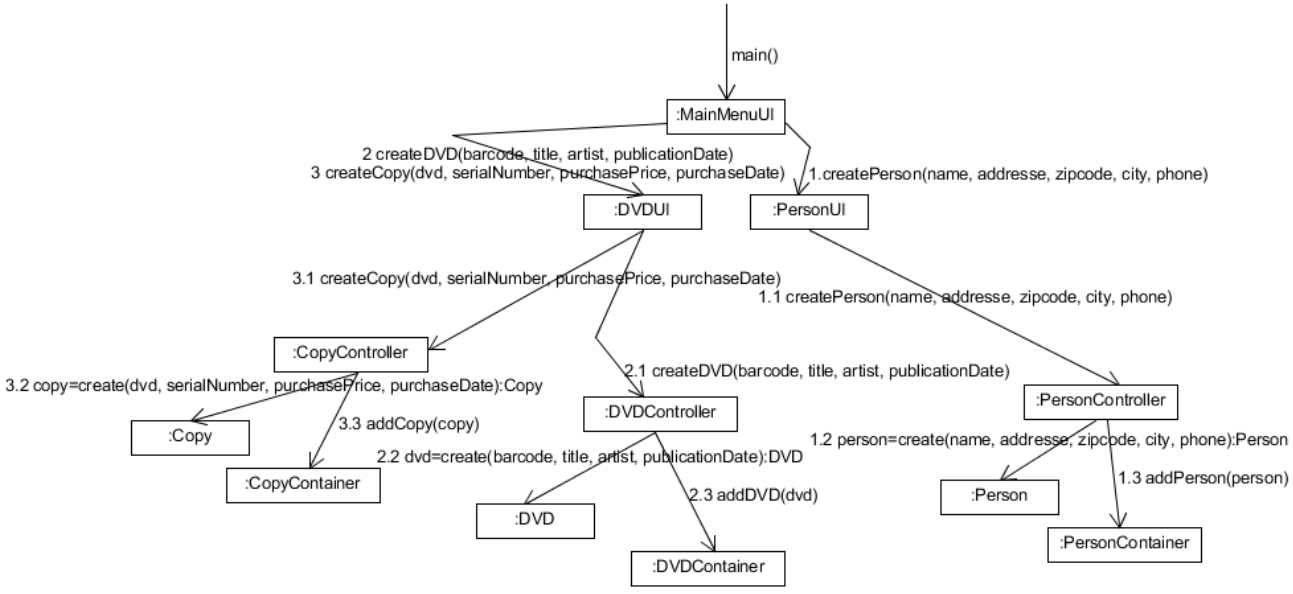
Her ses SSD diagrammet, som viser hvordan en DVD og en kopi oprettes, tilgås, opdateres og kan slettes. Først benyttes createDVD, derefter bliver en DVD tilføjet til en ArrayList. Så bliver metoden createCopy benyttet, som laver en kopi af DVD'en. Så bliver kopien tilføjet til en ArrayList. Der er også lavet CRUD metoder, som tager et objekt ind, dette objekt er enten en DVD eller en kopi.

Operationskontrakt

Operation: createDVD() Operation: addDVD() Operation: createCopy() Operation: addCopy() Operation: getInfo() Operation: update() Operation: delete()
use case: Manage DVD - CRUD Præcondition: Der udkommer en ny dvd Postcondition: En kopi af DVD'en er blevet oprettet, opdateret og slettet

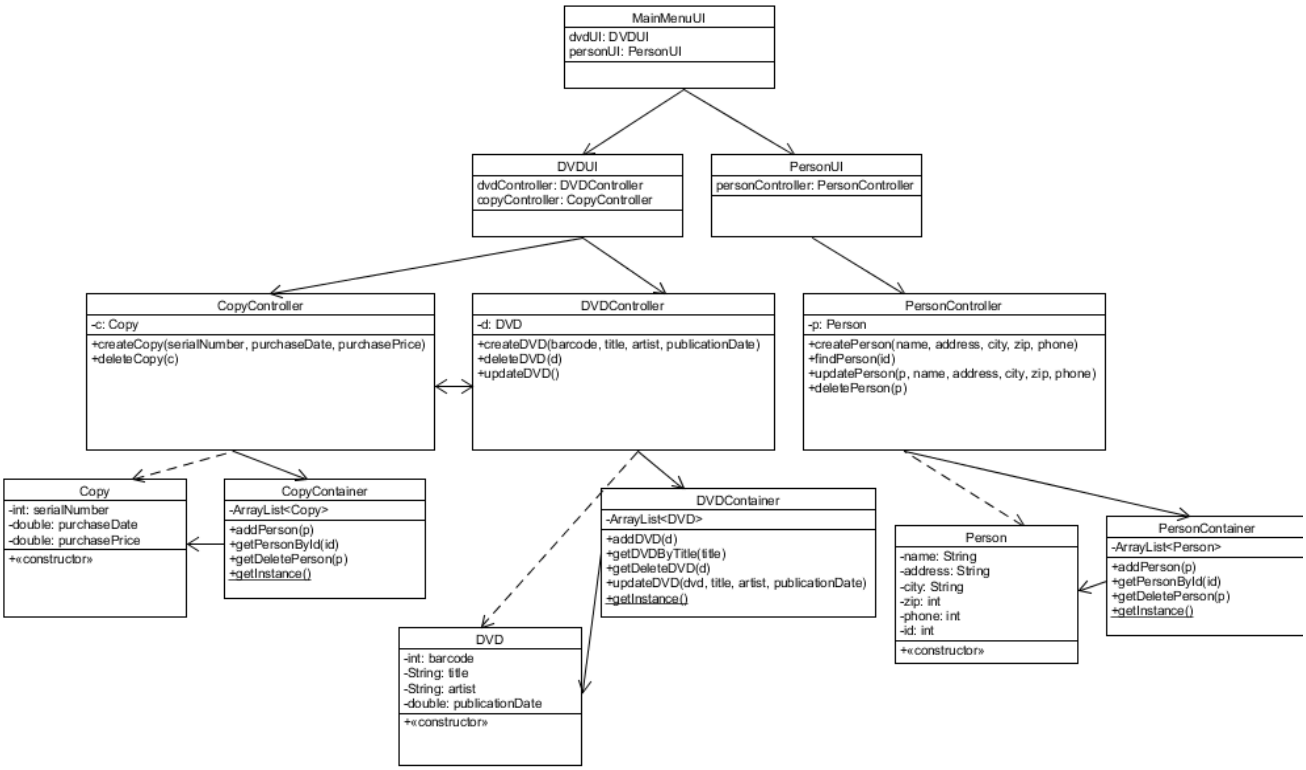
Dette er operationskontrakten, som giver et overblik over hvilke metoder der laver ændringer i systemet. Til "Manage DVD - CRUD", skal der laves CRUD.

Kommunikationsdiagram



Dette er udvidelsen af kommunikationsdiagrammet der er opgivet i opgaven, som har PersonUi, PersonController, PersonContainer og Person. Her er de blevet tilføjet en MainMenuUI, som holder styr på de forskellige UI's, så klassen UI ikke bliver for rodet. Der er også blevet lavet controllere, container og selve klassen, for både DVD og kopi.

Designklasse diagram



Til designklasse diagrammet er der blevet tilføjet DVD klasser og Copy klasser, som består af UI, controller, container og selve DVD samt Copy klassen. Klasser, metoder, synlighed og datatyper er også blevet tilføjet.

Delkonklusion

Det er nu muligt at, lave, redigere og slette en DVD, samt oprette kopier af DVD'en som også kan slettes. Det er dog stadig ikke muligt at låne kopier af DVD'er til en person i systemet, dette ville blive kigget på i den næste iteration.

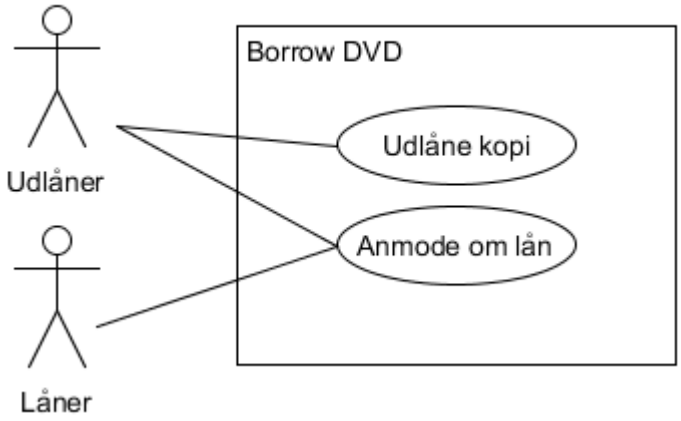
Iteration 3

Introduktion

Diagrammer

Use case

Her ses den komplekse use case “Borrow DVD”.

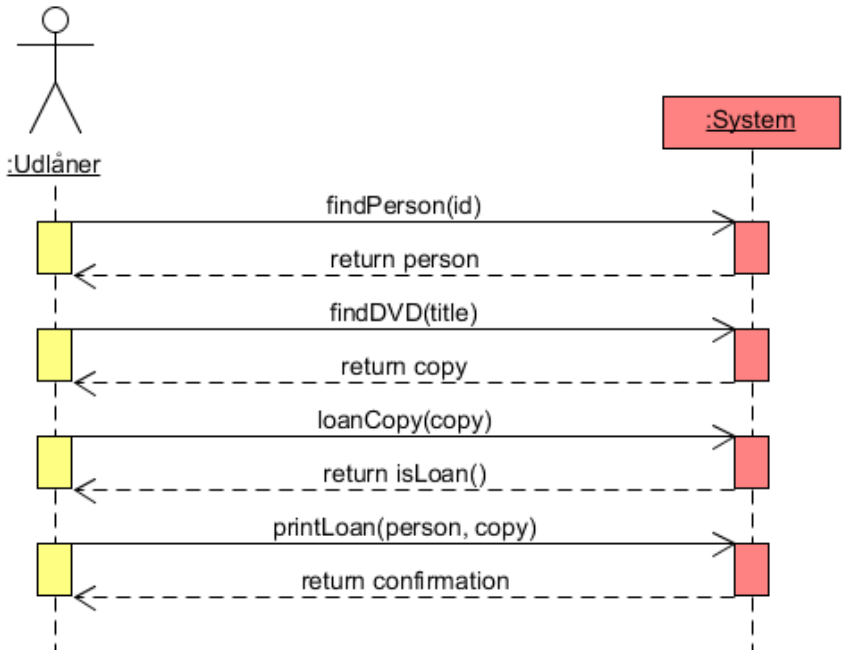


Fully dressed use case

Use case navn	Borrow DVD	
Aktører	Udlåner og låner	
Præbetingelser	Låner og kopi eksisterer	
Postbetingelser	Et lån er oprettet og associeret med låner og kopi, hvis den ønskede kopi er til stede	
Frekvens	0-1 gang om dagen	
Flow of events	Aktør	System

	1. En låner ønsker at låne en DVD	
	2. Udlåneren indtaster navn eller id	3. Systemet finder personen
	4. Låner fortæller hvilken DVD der ønskes, udlåneren indtaster oplysninger	5. Systemet returnerer informationer om tilgængelige kopier
	6. Låneren godkender lånet	7. Systemet registreret at lånet er lavet
Alternative flow	4. Ønskede dvd findes ikke	

SSD



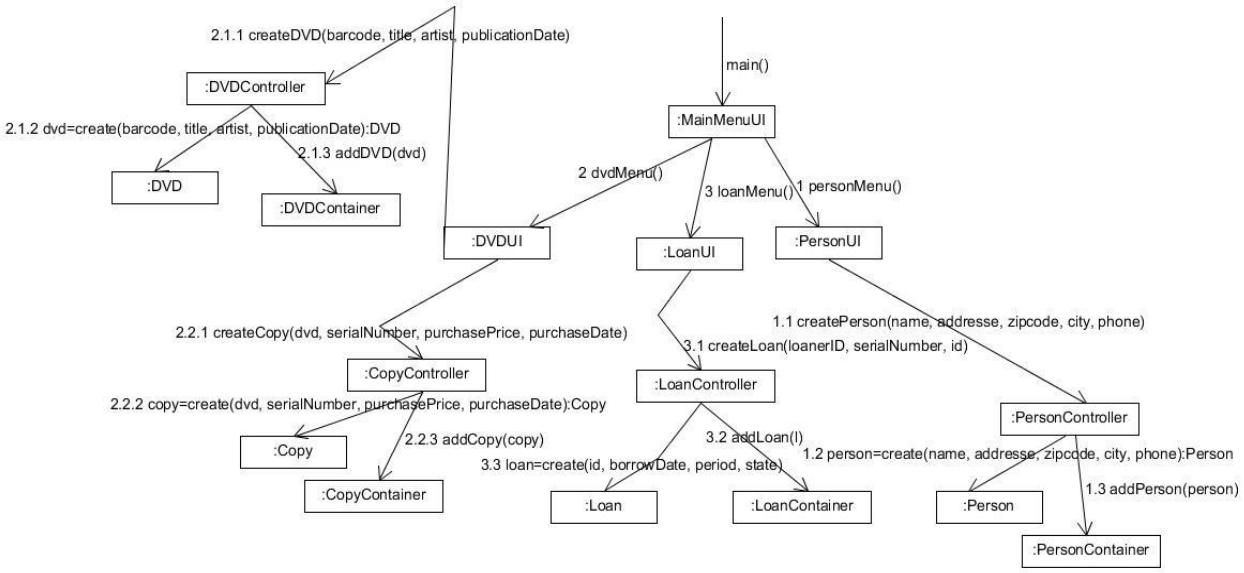
SSD diagrammet til use casen ”Borrow DVD”. Der skal findes en person via id, herefter bliver personen returneret. Så skal der findes en DVD via en titel, dernæst bliver der returneret kopierne af den givende DVD. Nu bliver der oprettet et lån, som tager kopien med sig og derefter bliver lånet printet, hvor den printer person og kopi.

Operationskontrakt

Operation: findPerson() Operation: findDVD() Operation: loanCopy() Operation: printLoan()
use case: Borrow DVD Præcondition: Låner og kopi eksisterer Postcondition: Et lån er oprettet og associeret med låner og kopi, hvis den ønskede kopi er til stede

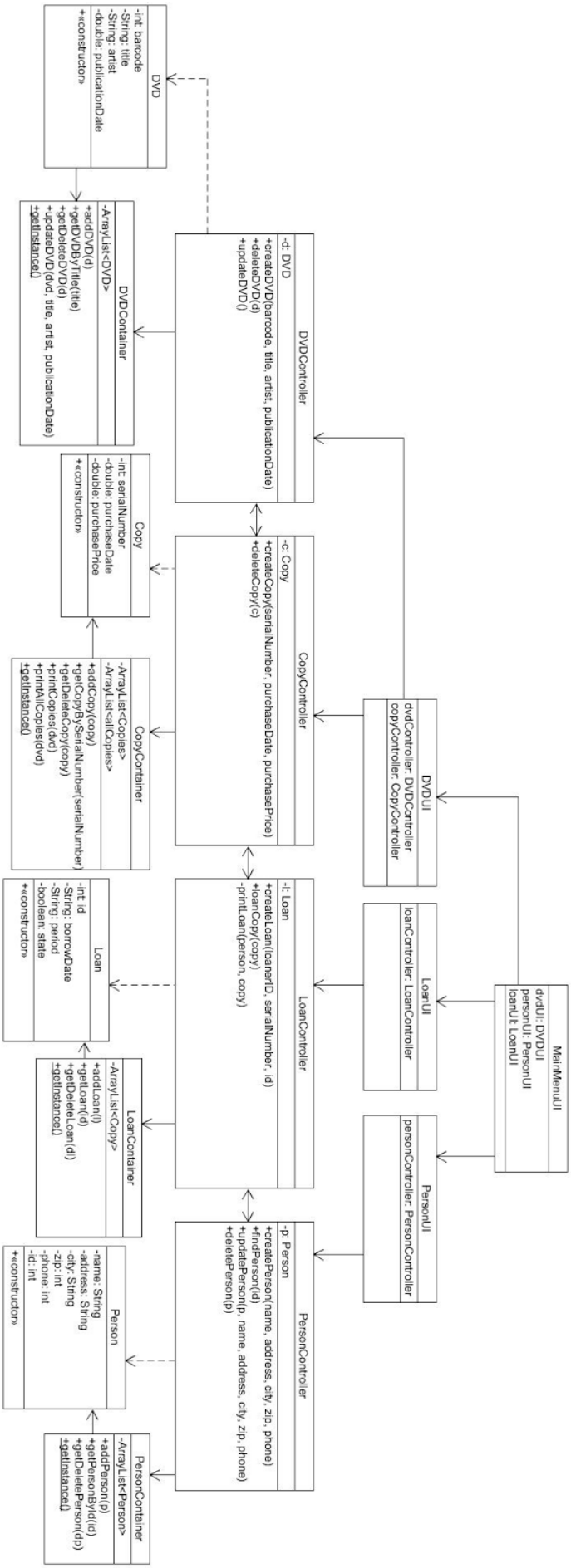
Ovenfor ses operationskontrakten, som giver et overblik over hvilke metoder der laver ændringer i systemet. Af use casen ”Borrow DVD”. Der skal findes en person, findes en DVD og derefter bliver den udlånt en kopi, hvorefter lånet bliver printet.

Kommunikations diagram



Der er blevet tilføjet lån, som har et UI, Controller, container og sig selv.

Designklasse diagram



Designklasse diagrammet er blevet udvidet med lån, som UI, Controllere, Container og selve klassen. Der er også blevet lavet forbindelse fra LoanControllere til PersonControllere og CopyControllere.

Delkonklusion

Det er nu muligt at en person, altså en låner kan ønske en kopi af en DVD, som de kan låne i en given periode på 7 dage. Det er dog endnu ikke muligt at returnere dette lån, hvilket der vil blive løst i næste iteration.

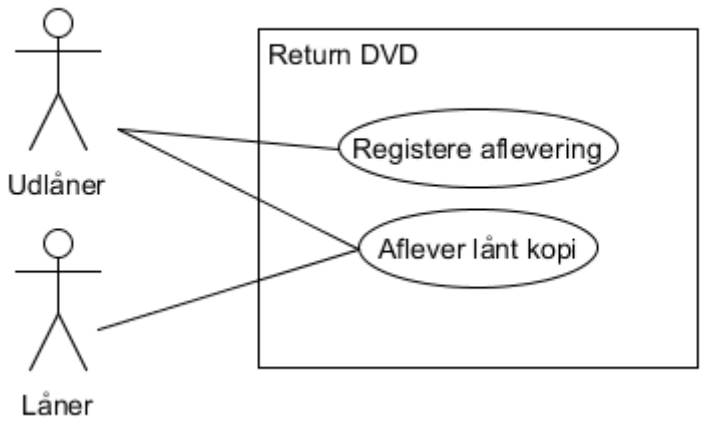
Iteration 4

Introduktion

Diagrammer

Use case

Dette er use casen "Return DVD".

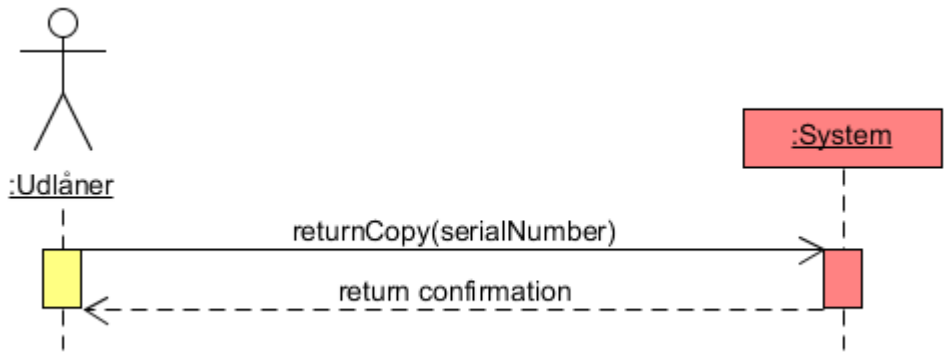


Fully dressed use case

Use case navn	Return DVD
Aktører	Udlåner og låner
Præbetingelser	Låner vil aflevere en kopi af DVD
Postbetingelser	Systemet registrerer at kopien er returneret

Frekvens	0-1 gang om dagen	
Flow of events	Aktør	System
	En låner ønsker at aflevere en kopi af DVD	
	2. Udlåneren tager imod kopien og registrere dens aflevering	3. Systemet registrerer at kopien er tilbage og er nu tilgængelig til at blive lånt ud igen
Alternative flow	3 Systemet genkender ikke kopien	

SSD



Der skal returneres en kopi, som bliver returneret via serienummeret for kopien, herefter bliver der returneret en bekræftelse.

Operationskontrakt

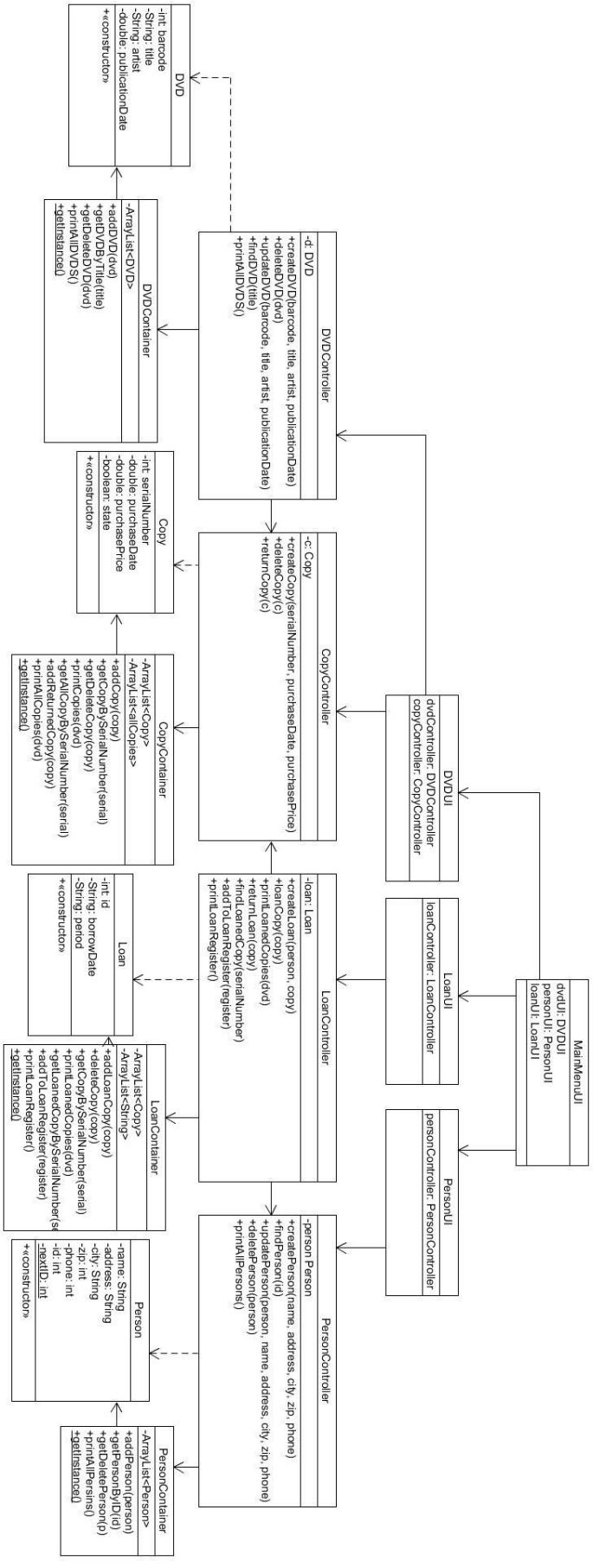
Operation: returnCopy()
use case: Return DVD Præcondition: En låner vil aflevere kopi af DVD Postcondition: Systemet registrerer at kopien er returneret

Dette er operationskontrakten, som giver et overblik over hvilke metoder der laver ændringer i systemet. Af use casen ”Return DVD”. Den ændring der sker er at en kopi af en DVD, bliver registeret som returneret.

Kommunikations diagram

Vi ændrede ikke noget i kommunikations diagrammet, da intet blev lavet, men bare returneret.

Designklasse diagram



Til iterations fire skulle der laves return metode, dette betød at der skulle tilføjes mange nye metoder i både controllerlag, containerlag og selve klasserne. Dette var nødvendigt da en løsning uden ikke var muligt på det givende tidspunkt.

Kode beskrivelse

Til Copy blev der lavet en singleton container. Containeren har to ArrayList; 'copies' der indeholder alle kopier der kan lånes ud hvorefter 'allCopies' indeholder alle kopier i systemet. Til at udarbejde en singleton er CopyContainer en statisk variabel i klassen selv og kan kun initialiseres igennem dens **getInstance()** metode.

```
public class CopyContainer
{
    private static CopyContainer instance;
    public static ArrayList<Copy> copies;
    public static ArrayList<Copy> allCopies;

    /**
     * Constructor for objects of class CourseContainer
     */
    private CopyContainer() {
        copies = new ArrayList<>();
        allCopies = new ArrayList<>();
    }

    public static CopyContainer getInstance() {
        if (instance == null) {
            instance = new CopyContainer();
        }
        return instance;
    }
```

Metoden **loanMenu()** i *LoanUI* der har til formål at sortere igennem et svar fra brugeren. Case 1 som vist nedenfor er for at oprette et lån. Til det skal der findes en person igennem metoden **findPerson()** der her ligges over i en instans variabel 'person' i klassen. Herefter spørger den om en DVD der her også ligges over i en instans variabel 'dvd' i klassen.

Når DVDen er fundet printer den alle ledige kopier af den DVD ud til brugeren. Herefter giver brugeren et serienummer der så laves et lån ud fra igennem **createLoan()**. Den printer herefter informationen omkring lånet ud til brugeren heriblandt datoen og hvilken DVD der blev lånt ud. Lånet bliver så 'added' til låne registeret ved at kalde **addToLoanRegister()** metoden.

```
public void loanMenu(){
    try{
        boolean exit = false;

        while(!exit) {
            int choice = writeLoanMenu();
            switch (choice) {

                case 1: //lån eksemplar til en person
                person = personUI.findPerson(); //find den person der skal låne et DVD eksemplar
                if(person != null){
                    System.out.println("Du fandt personen: " + person.getName() + " med id: " + person.getID());

                    System.out.println("Find DVD'en ved at indtaste DVD'ens titel");
                    dvd = dvdUI.findDVD(); //find DVD'en og dernæst eksemplaret der skal udlånes
                    if(dvd != null){
                        System.out.println("Du fandt DVD'en: " + dvd.getTitle());
                        dvdUI.printCopies(dvd);

                        System.out.println("Find eksemplaret ved at indtaste eksemplarets serienummer");
                        copy = dvdUI.findCopy();
                        if(copy != null){
                            System.out.println("Du fandt eksemplaret: " + copy.getSerialNumber());
```

```
                createLoan(); // udfør lånet
                if(loan != null){
                    System.out.println("DVD'en med titlen: " + dvd.getTitle() + " og med serienummeret: " +
                        copy.getSerialNumber() + " blev lånt til: " + person.getName() + " med id: " + person.getID() +
                            " DVD'en blev lånt den " + loan.getBorrowDate() + " og skal afleveres senest den " + loan.getPeriod() + "

                    String register = " DVD'en med titlen: " + dvd.getTitle() + " og med serienummeret: " +
                        copy.getSerialNumber() + " blev lånt til: " + person.getName() + " med id: " + person.getID() +
                            " DVD'en blev lånt den " + loan.getBorrowDate() + " og skal afleveres senest den " + loan.getPeriod() + "

                    addToLoanRegister(register); //tilføj lån til register.
                }
                else{
                    System.out.println("Lånet er ikke muligt!");
                }

            }
            else{
                System.out.println("Eksemplaret findes ikke!");
            }
        }
        else{
            System.out.println("DVD'en findes ikke!");
        }
    }
```

Test beskrivelse

Iteration 1

Til første iteration er der udarbejdet unit tests til *Person* og *PersonContainer* klasserne. I *Person* klasssen er der lavet tests på **getName()**, **setName()** og **getID()** metoderne. Grunden til at der ikke er lavet unit tests på alle 'set' metoderne i *Person* klassen er grundet deres identiske logik og da der ikke er udarbejdet 'get' metoder til alle instansvariablerne i *Person*, som ville være nødvendige for at tilgå deres respektive variabel. Det er også ikke ikke en del af vores kode standard at lave tests på metoder uden nogen dybere logik.

For *PersonContainer* klassen er der udarbejdet unit tests til **getInstance()**, **getPersonByID()**, **getDeletePerson()** metoderne. Til testen af **getPersonByID()** er der brugt **addPerson()** metoden fra *PersonContainer* klassen, men da den kun tilføjer en person til containerens liste var det ikke nødvendigt at lave en test, men derimod benytte den i førnævnte testmetode.

Iteration 2

I anden iteration er er udarbejdet unit tests til *DVD* og *DVDContainer* klasserne. For *DVD* klassen er der lavet tests på metoderne **getTitle()** og **setTitle()** da klassen kun indeholder metoder med samme funktionaliteter for deres respektive instans variabler.
For *DVDContainer* er der lavet unit tests til **getInstance()**, **getDVDByTitle()** og **getDeleteDVD()**. Som i *PersonContainerTest* klassen har de samme funktionaliteter med forskellene liggende i hvilke elementer de bearbejder.

Iteration 3 & 4

I disse iterationer blev der udarbejdet test klasser til *Loan*, *LoanContainer* og *CopyContainer*.
CopyContainerTest klassen har tests på metoderne **getAllCopyBySerialNumber()**, **getCopyBySeriaNumber()** og **getDeleteCopy()**.
LoanTest klassen tester metoden **setPeriod()** og den formatter der er implementeret igennem **DateTimeFormatter** i Java.
LoanContainerTest klassen tester metoderne **getCopyBySerialNumber()** og **deleteCopy()**.
Test klasserne udarbejdet i iteration 1 og 2 der benyttede *CopyContainer* instansen, der er en singleton, gav nogle komplikationer til udarbejdelsen af test klassen til *LoanContainer og CopyContainer*. Komplikationen indebar at da vi arbejder med en singleton der kun har dens specifikke liste af kopier og så har to test klasser er 'adder' nye kopier hver især så har listen stadig de tidligere 'added' kopier i andre testklasser. Derfor var der behov for i **testGetAllCopyBySerialNumber()** test metoden i *CopyContainerTest* klassen, at bruge højere 'serialNumber's' end ellers. Det samme for **testGetCopyBySerialNumber()** i *LoanContainerTest* klassen.

Delkonklusion

Efter iteration fire er det nu muligt at aflevere den lånte kopi af DVD'en.

Evaluering af gruppearbejde

Vores gruppearbejde har været meget vellidt blandt gruppens medlemmer. Der har været meget fokus på, at alle skulle have muligheden for at sige, hvad den enkelte tænker og stille spørgsmål til de ting, der er blevet udarbejdet. Hvis der er blevet stillet spørgsmål, har der ikke været nogen sure opstød eller følelser med ens mening eller spørgsmål ikke har været velkommen. Der er blevet lyttet til alle i gruppen og der er blevet udarbejdet diagrammer, programmering samt rapport i samarbejdet med hinanden og der ikke blevet opdelt opgaver i blandt gruppens medlemmer. Det primære formål har været at alle skal lære og alle skal være med.

Udover dette, har der ikke været nogen negative diskussioner i gruppen, men mange konstruktive diskussioner, hvor der er blevet delt viden og erfaringer som har gjort at det har været nemmere at komme frem til et godt resultat og der ikke har været nogen tidspunkter, hvor gruppen har haft følelsen af vi ikke har opnået de mål, der var blevet sat for dagen.

Vi er derfor enige om i gruppen, vi har haft et godt samarbejde og noget de målsætninger gruppen havde sat i begyndelsen.

Konklusion.

Det blev først muligt at lave CRUD på personer, derefter blev det muligt at lave CRUD på DVD'er samt kopier. Systemet kan også oprette et lån og returnere lånet. Det blev også muligt at se hvilken person der havde lånt kopien og hvornår lånet skulle afleveres samt om lånet overholdte afleveringsfristen. Ud fra systemet har gruppen opnået forståelse for 3-lagsarkitekturen og de komplikationer der kan fremkomme hvis man bryder arkitekturstrukturen.

Bilag

Gruppe kontrakt

Hvad er vigtigt for gruppearbejdet

Det er vigtigt at der blive givet plads til alle, hvor den enkelte har mulighed for faglig samt personlig udvikling. Målet er at løfte opgaven i fællesskab, hvilket vi kan opnå igennem et godt samarbejde i gruppen, hvor der på forhånd er blevet sat nogen krav til den enkelte gruppemedlem og hvordan det vil blive håndteret, hvis et gruppemedlem ikke overholder de aftaler, der er blevet udarbejdet i gruppen. Dette indebærer blandt andet mødestabilitet, hjemmearbejde og hvordan vi har tænkt os, at løse eventuelle problemer, som kan opstå i gruppen.

Kontraktbrud og konsekvenser

Hvis et gruppemedlem ikke overholder de krav og betingelser, som der er blevet beskrevet i kontrakten, vil resten af gruppen i første omgang, have muligheden for at give en advarsel om at en person vil blive udmeldt af gruppen, hvis personen ikke begynder at overholde de punkter, der er blevet beskrevet i kontrakten. Hvis gruppemedlemmet endnu ikke retter sig ind og bliver ved med at lave brud på gruppekontrakten samt de betingelser der er blevet stillet i forbindelse med advarslen. Har gruppen mulighed for at udmelde gruppemedlemmet, i samarbejde med kontaktpersonen, som er blevet tildelt gruppen.

Hvad gruppen glæder sig til

Gruppen glæder sig til at kunne arbejde med den teori, som der blevet undervist og bruge det i praktisk til at løse den givet opgave. At kunne tilegne sig mere viden og erfaring indenfor vores studieområde, men også blive bedre til at samarbejde, så alle i gruppen får det maksimale ud af projektet og ender med en god mavefornemmelse, med de har ydet deres bedste og har haft en lærerig proces.

Krav og betingelser

1) Alle gruppemedlemmer skal møde, til den aftalte mødetid
    a. Der skal gives besked, hvis man er forsinket.
    b. Der skal gives begrundelse, hvis man bliver forhindret i at møde.
    c. Mødetiden skal gives mindst 24 timer før.
2) Mødestedet
    a. Mødestedet skal besluttes mindst 12 timer før mødetidspunktet.
    b. Mødesteder skal aftales i gruppen, hvis det ikke er forud bestemt.
    c. Mødestedet skal være retfærdig, for alle gruppemedlemmer. (Afstand og transport).
3) Fri og frihed

a. Mellem klokken 8:30 – 16 har gruppen arbejdstid.
    i. Gruppen har mulighed for at ændre tidspunkterne med varsel.
    ii. Gruppemedlem har mulighed for at gå tidligere, med god begrundelse. (F.eks. arbejde eller lignende).
b. Hjemmearbejde
    i. Der skal gives hjemmearbejde i god tid, så det enkelte medlem har mulighed for at tilrettelægge sin fritid.
    ii. Gruppemedlemmer har ikke pligt til at udføre hjemmearbejde, hvis der bliver givet en god begrundelse for, hvorfor medlemmet ikke har mulighed for at udføre det.
    iii. Hjemmearbejde skal gives med deadlines og skal være retfærdigt.
c. Frihed
    i. Det skal være muligt for alle medlemmer at have et personligt liv, ved siden af studiet. (Dette punkt er ugyldigt tæt på deadlines, hvis gruppen er bagud for tidsplanen).

4) Aftaler
a. Hvis der bliver lavet aftaler i gruppen, skal disse overholdes.
    i. Aftaler skal være fair, for alle partner
    ii. Gyldige aftaler, skal være på skrift og godkendes.

Underskrifter

_____ _____

Daniel Pedersen Dato: DD-MM-YYYY

_____ _____

Jonas Weile Dato: DD-MM-YYYY

_____ _____

Peter Madsen Dato: DD-MM-YYYY

_____	_____
Martin Lundgren	Dato: DD-MM-YYYY
_____	_____
Jacob Thomsen	Dato: DD-MM-YYYY