DVD lånesystem

Miniprojekt: Design
Datamatiker, 1. semester

Dato: 09-11-2018

University College Nordjylland

Teknologi og business
Erhvervsakademiuddannelsen Datamatiker

Udvikling af DVD lånesystem
Hold 6
Dmab0918

Projektdeltagere:

Dennis Rebbe Henriksen
Caspar Emil Jensen
Tobias Rye Justesen
Jimmy Poulsen
Jeppe Just Svendsen

Vejledere:

Anita L. Clemmensen
István Knoll

Afleveringsdato: 09-11-2018

_____

Underskrift

Indholdsfortegnelse

Indledning

Vi har fået til opgave at udvikle et DVD-lånesystem. Kravspecifikationerne for systemet har været forudbestemte og er som følger: det skal have et funktionelt, dynamisk register, hvor DVD-film, og informationer herom, samt eventuelle kopier kan lagres, ændres og fjernes. Ligeledes skal der være et kartotek af registrerede lånere, som kan ændres løbende. Endeligt skal det være muligt at se, hvem der har lånt hvad; hvornår en given film skal returneres samt eventuel dato for returnering.

Rapporten vil løbende gennemgå arbejdsprocessen for udviklingen af førnævnte system ved hjælp af beskrivelse af kode og arkitektur samt unit tests af de forskellige klasser.
Medfølgende er desuden en domænemodel, der giver et overordnet syn på systemets opbygning. Denne bliver opfulgt af fully-dressed use case beskrivelser samt system sekvens diagrammer og operations-kontrakter, og endelig et komplet design klassediagram, der demonstrerer det komplette system og dets funktioner.

Kode konvention

1. Pakker og import

De første linjer i koden består af import statements. Disse muliggør brugen af eks. ArrayList, HashMap og Scanner. Desuden importeres også TUI-, controller-, og model-packages alt efter behov.

E.g. *package tui;*
import java.util.Scanner;

2. Klasse og brugerflade deklaration

Følgende er den kronologiske rækkefølge af deklarationer i koden:

1. Klasse og brugerflade deklaration
2. Statiske klasse variabler
3. Instans variabler
4. Constructors
5. Metoder

3. Generel opstilling

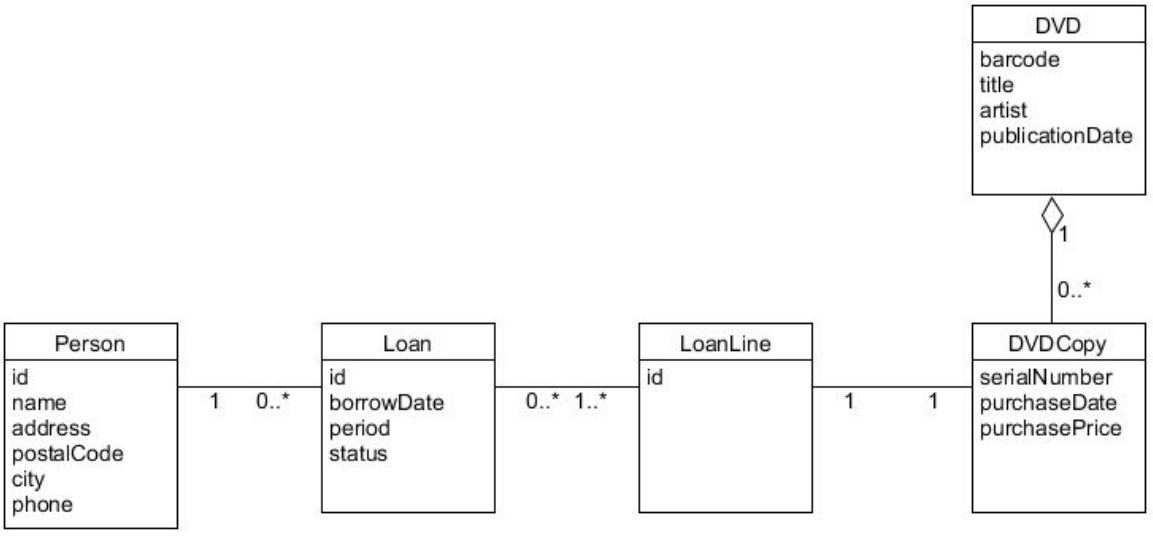
1. Indrykning foregår ved hjælp af tabulatortasten, som er forudindstillet til 4 indryk med mellemrumstasten.
2. Mellemrum efter komma og før en operatør.
3. I tilfælde af at en linje bliver for lang, da brydes denne med et afsnit og fortsættes i samme niveau, som den forrige linje - evt. med et enkelt indryk pr. linje.

4. Placering og formatering

1. Krøllet parentes (start) placeres slutteligt på samme linje som erklæringen.
2. Lokale variabler deklareres i starten af kode blokken for hvert enkelt erklæring og gentages ikke længere inde i koden.
3. Krøllet parentes (slut) placeres nedenunder blokkens afslutning.
4. Undtagelsesvist foregår deklarationen af lokal variablen, i et for-loop, på samme linje som erklæringen.
5. Ingen luft mellem metodenavn og start parentes.
6. Én deklaration pr. linje.
7. Ved kun én betingelse i et if-else statement udelades krøllede parenteser, såfremt linjen er indenfor læselig grænse.

Domænemodel

Domænemodellen som vi har lavet, ses på bilag 2.1. Domænemodellen viser hvordan klasserne hænger sammen. Vi har valgt at tilføje en LoanLine til den domænemodel vi fik udleveret. Dette skyldes, at man som udgangspunkt ikke kan have flere DVD'er i et lån, såfremt man ikke implementerer LoanLine. Et lån har altså én eller flere låne linjer. Derudover har vi ændret State til Status, for at fortælle hvorvidt lånet er aktivt, inaktivt eller over afleveringsdatoen, samt tilføjet id til Person, som vi bruger som key i vores hashMap.
I Modellen viser vi at Loan er associeret med Person. Loan har en Person og Person kan have flere Loans. Loan er associeret med LoanLine. Loan kan have flere LoanLines og LoanLine tilhører ét Loan. LoanLine og DVDCopy er associeret med hinanden og kan kun have en af hinanden knyttet til sig. DVDCopy indgår i DVD gennem aggregeringen mellem de to klasser. DVDCopy kan have en DVD og DVD kan have 0 til mange DVDCopy.



Bilag 2.1

Fully Dressed Use-Cases

Nedenfor ses en *Fully Dressed Use Case* for “Lend DVD”. Dette er en mere udpenslende variant af en *Brief Case.*

I nedenstående eksempel vil Joe, som aktør, gerne låne en DVD-film (Lend DVD). Før at dette kan lade sig gøre, kræves det, at Person- og DVDCopy-objekter er oprettet. Dernæst forløber begivenhederne som angivet i “Flow of Events”, og afslutningvist skal postbetingelserne være opfyldt - altså, et lån skal være blevet oprettet og associeret til et Person- og DVDCopy-objekt.

På diagrammet kunne være inkluderet “Alternative Flows”, hvor eventuelle alternative udfald kunne være nævnt. Et eksempel på dette: “2a. ID eksisterer ikke.”

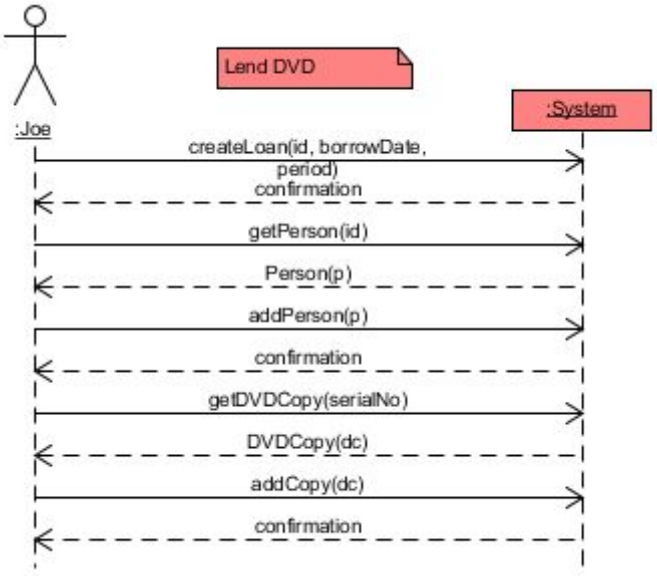
Use case navn	Lend DVD	
Aktør	Joe	
Prebetingelser	Person og DVDCopy er oprettet	
Postbetingelser	Et lån er oprettet og associeret med en Perosn og DVDCopy. DVDCopy er tilgængelig	
Frekvens	0-20 gange	
Flow of Events	**Aktørhandling**	**System svar**
	1. Joe vil låne en DVD	
	2. Joe indtaster sit ID	3. Systemet finder hans personoplysninger
	4. Joe indtaster hvilken DVDCopy han vil låne	5. Systemet returnerer DVDCopy-informationen
	6. Joe afslutter lånet	7. Systemet udåner DVDCopyen til Person og returnerer at lånet er

Se yderligere Fully Dressed Use Cases i bilag 1.4 - 1.7.

System Sekvens Diagram

I vores system sekvens diagrammer vil vi vise hvordan lånesystemet håndterer use casen Lend DVD. De resterende SSD'er er i bilag 1.3, men vi tager udgangspunkt i Lend DVD, da det er den med mest komplexitet.

- Lend DVD
  Brugeren påbegynder at oprette et nyt lån. Her skal der indtastes personens ID. Dernæst startes et flow, hvor der kan tilføjes DVD'er. Dette foregår ved at indtaste barcode på DVD'en, efterfulgt af serialNo på DVDCopy.



-

- Se det fulde SSD diagram i bilag 1.3.

Operations kontrakter

Disse er mere dybdegående og tekniske end Use Cases og domænemodellen, og i forhold til System Sekvens Diagrammerne - som viser begivenheder i systemet - så bruger vi operations kontrakter til at vise beskrivende ændringer i objekterne.

Operation: createDVD (barcode, title, artist, publicationDate)
Use case: Manage DVD – CRUD
Pre-betingelser: DVDContainer eksisterer
Post-betingelser:

- Et DVD-objekt *d* er blevet oprettet
- *d* er associeret til DVDContainer
- d.barcode er blevet til barcode,
  d.title er blevet til title,
  d.artist er blevet til artist,
  d.publicationDate er blevet til publicationDate

Her ses en operations kontrakt for metoden “createDVD”, som tager parametrene barcode, title, artist, publicationDate. Den nedstammer fra Use Casen “Manage DVD - CRUD.
Før operationen kan udføres, er det påkrævet, at en Container af typen DVDContainer eksisterer. Efter endt udførelse skal et DVD-objekt af typen “d” være blevet oprettet og associeret til DVDContainer, og har desuden taget parameterværdierne.

Se yderligere operations kontrakter i bilag 2.2 - 2.8.

Interaktionsdiagram - Lend Dvd

Med udgangspunkt i tre-lags arkitekturen - har vi udfærdiget et interaktionsdiagram, som ses på bilag 1.8 over use case “Lend DVD”, med et interaktions diagram søger vi at klarlægge de interaktioner, systemets forskellige klasser har med hinanden under udførelsen af denne use case.

- Systemet starter i < Loan UI>, her spørges der hvilken person som vi tillægges lånet samt hvilken dvd(er) der skal lånes. Dette gøres ved at hente information fra hhv. <PersonContainer> klassen, samt <DVDContainer> klassen. Dernæst vælges en <DVDCopy> af den pågældende dvd som skal lånes - dette gøres ud fra serial number. Det er et unikt nummer som kan identificere hvilken entitet vi har af samme type dvd.
- Systemet går dernæst videre, og der skal nu oprettes et lån, som bliver gemt i <Loan containter> klassen. Her oprettes et <Loan> objekt som tillægges værdierne(id, borrowDate, period samt status). Vi har også de resterede CRUD metoder. Read = FindLoan, Update = updateLoan og Delete = deleteLoan.

Interaktionsdiagram - Manage Dvd

Med samme udgangspunkt som i bilag 1.8 har vi lavet et interaktionsdiagram som ses på bilag 1.9 omkring use case “manage DVD” og “manage DVDCopy”. Som igen skal vise systemets interaktion imellem klasser i systemet.

- Systemet starter i DVDUI, her bliver brugeren spurgt efter hvilken funktion der skal bruges. Brugeren får mulighed for at oprette en DVD, Finde DVD, opdaterer DVD, slette DVD, oprette DVDCopy, finde DVDCopy, opdaterer DVDCopy og slette DVDCopy.
- Alle disse funktioner foregår i DVDController klassen. DVDController har også funktionen til at tilføje DVD til DVDContaineren og DVDCopy til DVD.

Design Klassediagram

På baggrund af vores øvrige modeller har vi lavet et design klassediagram som ses på bilag 2.9 - som skal ses som et diagram der kan give det store overblik over samtlige klasser vi har i vores system. Vi kan yderligere også se referencer til andre klasser, metoder samt deres synlighed. Tre-lags arkitekturen kommer til udtryk i diagrammet, ved at vi har UI, Controller samt Container. Vi har valgt at opbygge vores system, med et MainUI, der kan tilgå PersonUI, LoanUI og DVDUI. det giver den fordel at man benytter sig af konceptet modularisering. PersonUI tilgår så PersonController, som så tilgår PersonContainer. Samme struktur gælder for Loan

og DVD sektionen. Det der adskiller DVD sektionen er at vi har benyttet os af en multipliceret klasse, som vi har kaldt DVDCopy. Det er en klasse som holder informationerne om de forskellige kopier vi kan have af den samme type dvd. De er kendetegnet ved at have unikke serienumre. Mens DVD klassen, er adskiller sig ved at have en stregkode(barcode). Eks. På den måde kan man have 3 udgaver af Klovn The Movie. En der er slidt, en der er lettere brugt og en der er helt ny.

Vi har lavet en reference mellem Loan og Person, samt Loan og DVDCopy. Det giver den mulighed at når man skal oprette et nyt Lån, kan man koble en person og en DVDCopy på lånet. Så man kan se hvem der har lånt hvad.

Arkitektur

Vi har lavet et 3-lags arkitektur, bestående af et præsentationslag, kontrollag og modellag. Præsentationslaget kaldes UI, user interface. Dette lags formål er at sende systemhændelser ned igennem lagene, som sker ved interaktionen mellem aktøren og grænsefladen. Kontrollaget har til formål at afvikle use cases, f.eks. Lend DVD. Modellaget er der vi har vores domæneklasser som f.eks. Person og DVD samt deres containerklasser. Containerklassens formål er at samle diverse objekter, f.eks i en arraylist som for DVD eller HashMap for Person.

Refleksion

Gennem opgaven har været i dybden med både programmering- og systemudviklings processen. Vi har formået at kreere et fejlfrit system som kan udleje film, samt opbevare data og informationer om en given person/film. Vi har samtidig også været meget omkring udviklingen af systemet gennem vores systemudviklingsmodeller, og skabt os et overblik over hvad vi skulle ende ud med at have i programmet, og hvilke krav der skulle opfyldes. Overordnet er projektet gået rigtig godt og gruppearbejdet har fungeret godt. Vi har overholdt tidsplanen og fået opgaverne lavet til tiden.

Kilder

Litteratur

[Larman] kap 10, s. 173-180 (system sekvens diagram)
[Larman] kap 11, s. 189-191 (operations kontrakt)

Web

Dictionary. (2018, 11 09). Retrieved from Dictionary: ”
(https://www.dictionary.com/browse/modularization)

PDF

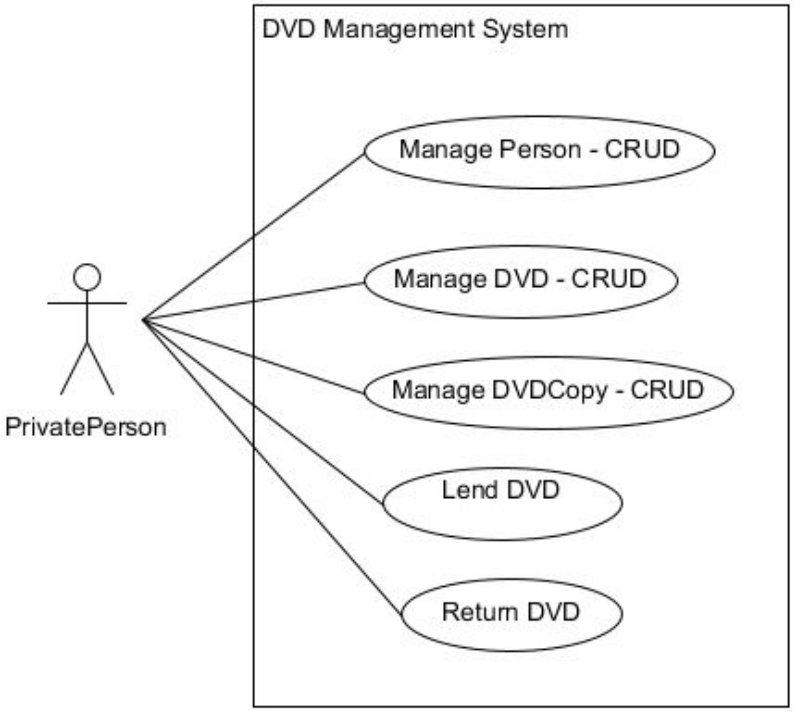
MiniProjectDesign_dmab0918.pdf
VersionControl_(lang)_dmab0918.pdf
VejledningRapportskrivning.pdf
codeconventions-150003.pdf

Ordliste

Modularisering:
Definition: “ at danne eller organisere i moduler, for fleksibilitet.” (Dictionary, 2018)

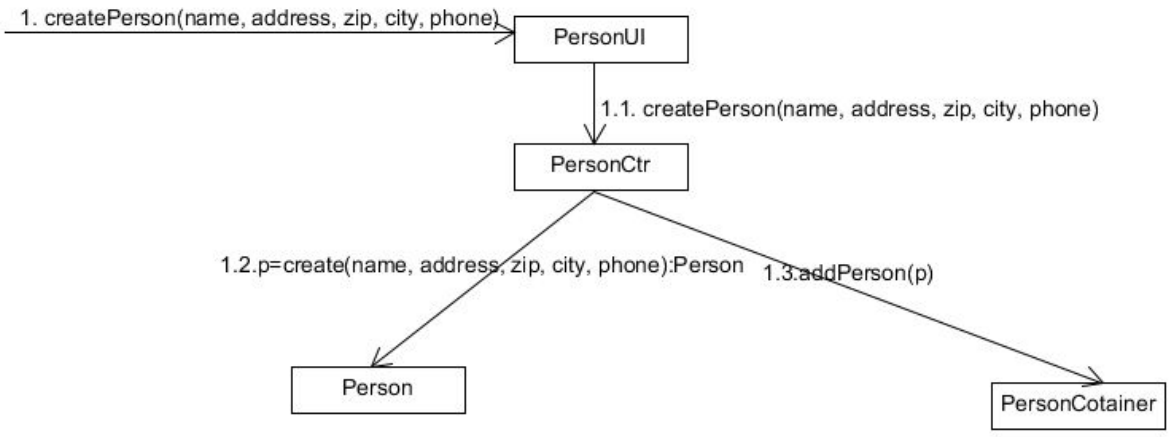
Bilag

1.1

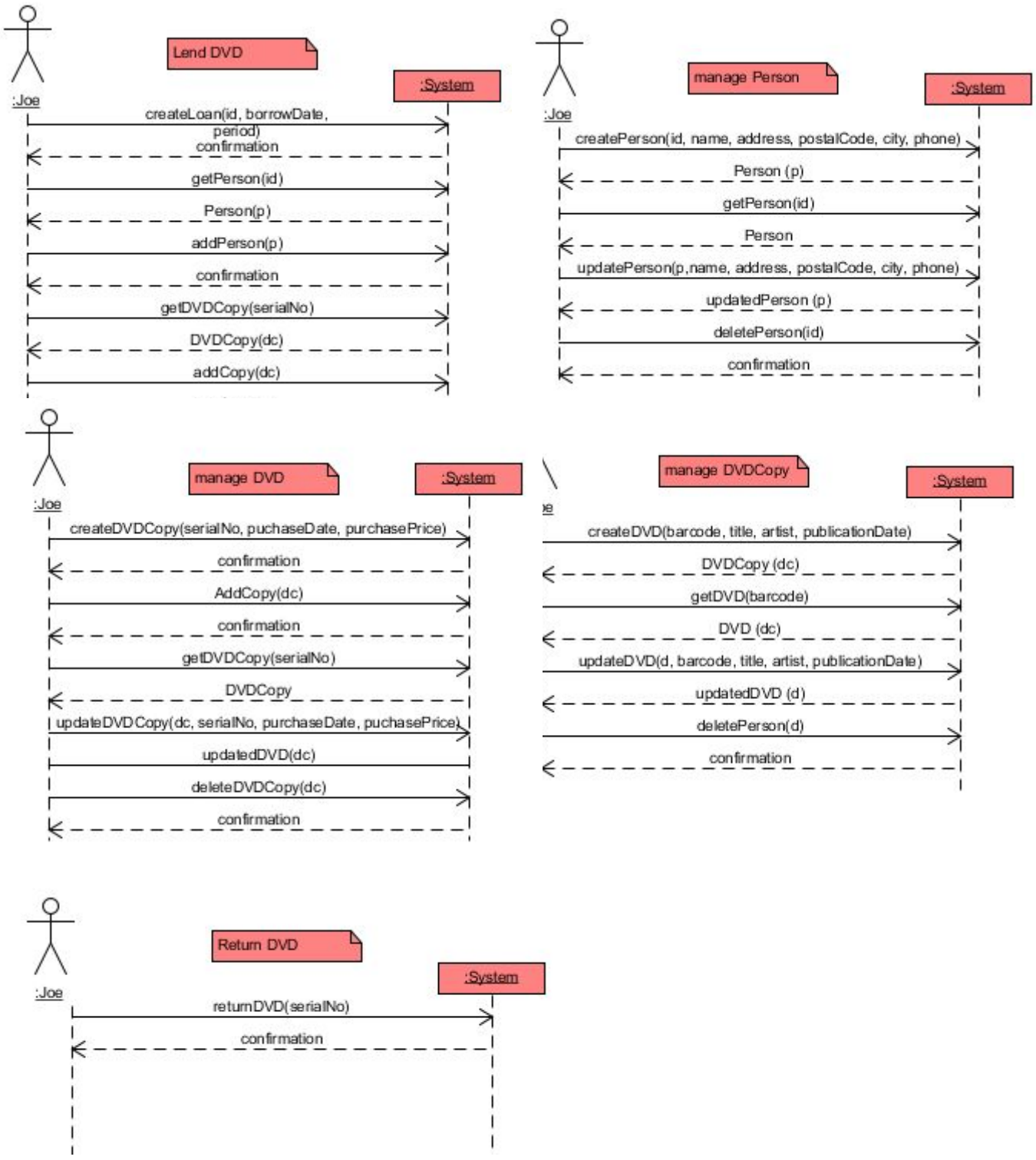


Ud fra den udleverede Use case har vi tilføjet ´Manage DVDCopy, så den er tilpasset til vores udarbejdelse.

1.2


1. createPerson(name, address, zip, city, phone)
PersonUI
1.1. createPerson(name, address, zip, city, phone)
PersonCtr
1.2.p=create(name, address, zip, city, phone):Person
1.3.addPerson(p)
Person
PersonCotainer

1.3


Lend DVD
:Joe
:System
createLoan(id, borrowDate, period)
confirmation
getPerson(id)
Person(p)
addPerson(p)
confirmation
getDVDCopy(serialNo)
DVDCopy(dc)
addCopy(dc)
manage Person
:Joe
:System
createPerson(id, name, address, postalCode, city, phone)
Person (p)
getPerson(id)
Person
updatePerson(p,name, address, postalCode, city, phone)
updatedPerson (p)
deletePerson(id)
confirmation
manage DVD
:Joe
:System
createDVDCopy(serialNo, puchaseDate, purchasePrice)
confirmation
AddCopy(dc)
confirmation
getDVDCopy(serialNo)
DVDCopy
updateDVDCopy(dc, serialNo, purchaseDate, puchasePrice)
updatedDVD(dc)
deleteDVDCopy(dc)
confirmation
manage DVDCopy
:System
createDVD(barcode, title, artist, publicationDate)
DVDCopy (dc)
getDVD(barcode)
DVD (dc)
updateDVD(d, barcode, title, artist, publicationDate)
updatedDVD (d)
deletePerson(d)
confirmation
Return DVD
:Joe
:System
returnDVD(serialNo)
confirmation

1.4

Use case navn	Manage DVD - CRUD	
Aktør	Joe	
Prebetingelser	Opret ny DVD	
Postbetingelser	ny DVD oprettet	
Frekvens	0 -10 gange	
Flow of Events	**Aktørhandling**	**System svar**
	1. Joe åbner MainMenuUI	
	2. Joe opretter ny DVD	3. Systemet opretter ny DVD

Use case navn	Manage DVD - CRUD	
Aktør	Joe	
Prebetingelser	DVD er i systemet	
Postbetingelser	DVD fundet	
Frekvens	0 -10 gange	
Flow of Events	**Aktørhandling**	**System svar**
	1. Joe åbner MainMenuUI	
	2. Joe søger efter DVD via barcode	3. Systemet finder DVD'en

1.5

Use case navn	Manage DVD - CRUD	
Aktør	Joe	
Prebetingelser	DVD er i systemet	
Postbetingelser	DVD opdateret	
Frekvens	0 -5 gange	
Flow of Events	**Aktørhandling**	**System svar**
	1. Joe åbner MainMenuUI	
	2. Joe opdaterer DVD ved at søge efter dens barcode	3. Systemet finder DVD
	4. Joe indtaster ændringer	5. systemet tilføjer ændringerne til DVD'en

Use case navn	Manage DVD - CRUD	
Aktør	Joe	
Prebetingelser	DVD er i systemet	
Postbetingelser	DVD slettet	
Frekvens	0 -5 gange	
Flow of Events	**Aktørhandling**	**System svar**
	1. Joe åbner MainMenuUI	
	2. Joe sletter DVD ved at søge efter dens barcode	3. Systemet finder DVD og sletter DVD

1.6

Use case navn	Manage Copy - CRUD	
Aktør	Joe	
Prebetingelser	Opret ny DVDCopy	
Postbetingelser	ny DVDCopy oprettet	
Frekvens	0 -10 gange	
Flow of Events	**Aktørhandling**	**System svar**
	1. Joe åbner MainMenuUI	
	2. Joe opretter DVDCopy	3. Systemet opretter DVDCopy
	4. Joe tilføjer DVDCopy til DVD lager	5. Systemet adder DVDCopy til DVD's arrayliste

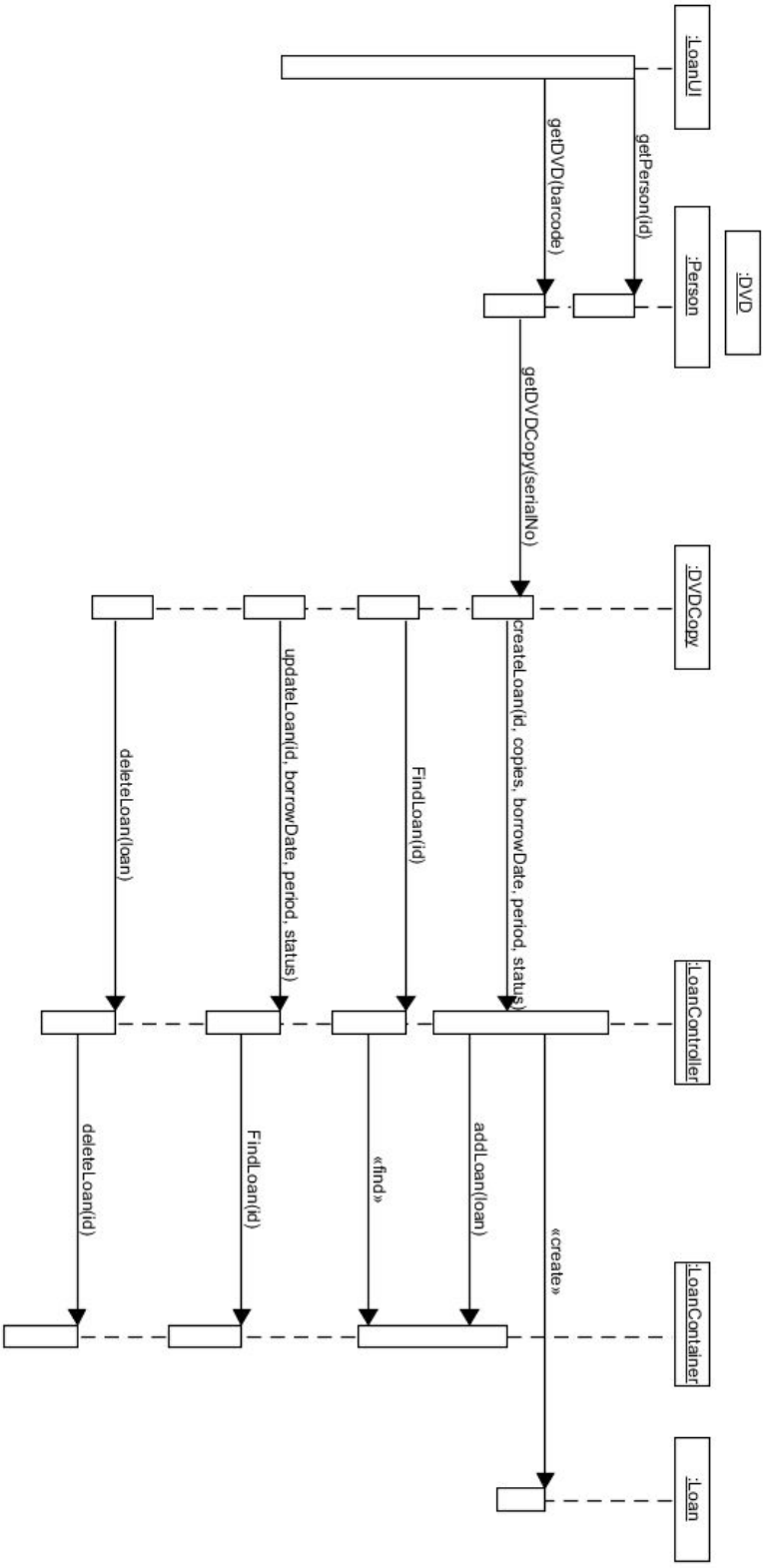
Use case navn	Manage Copy - CRUD	
Aktør	Joe	
Prebetingelser	DVDCopy findes	
Postbetingelser	DVDCopy fundet	
Frekvens	0-20 gange	
Flow of Events	**Aktørhandling**	**System svar**
	1. Joe åbner MainMenuUI	
	2. Joe søger efter DVDCopy via serialno.	3. Systemet finder DVDCopy'en

1.7

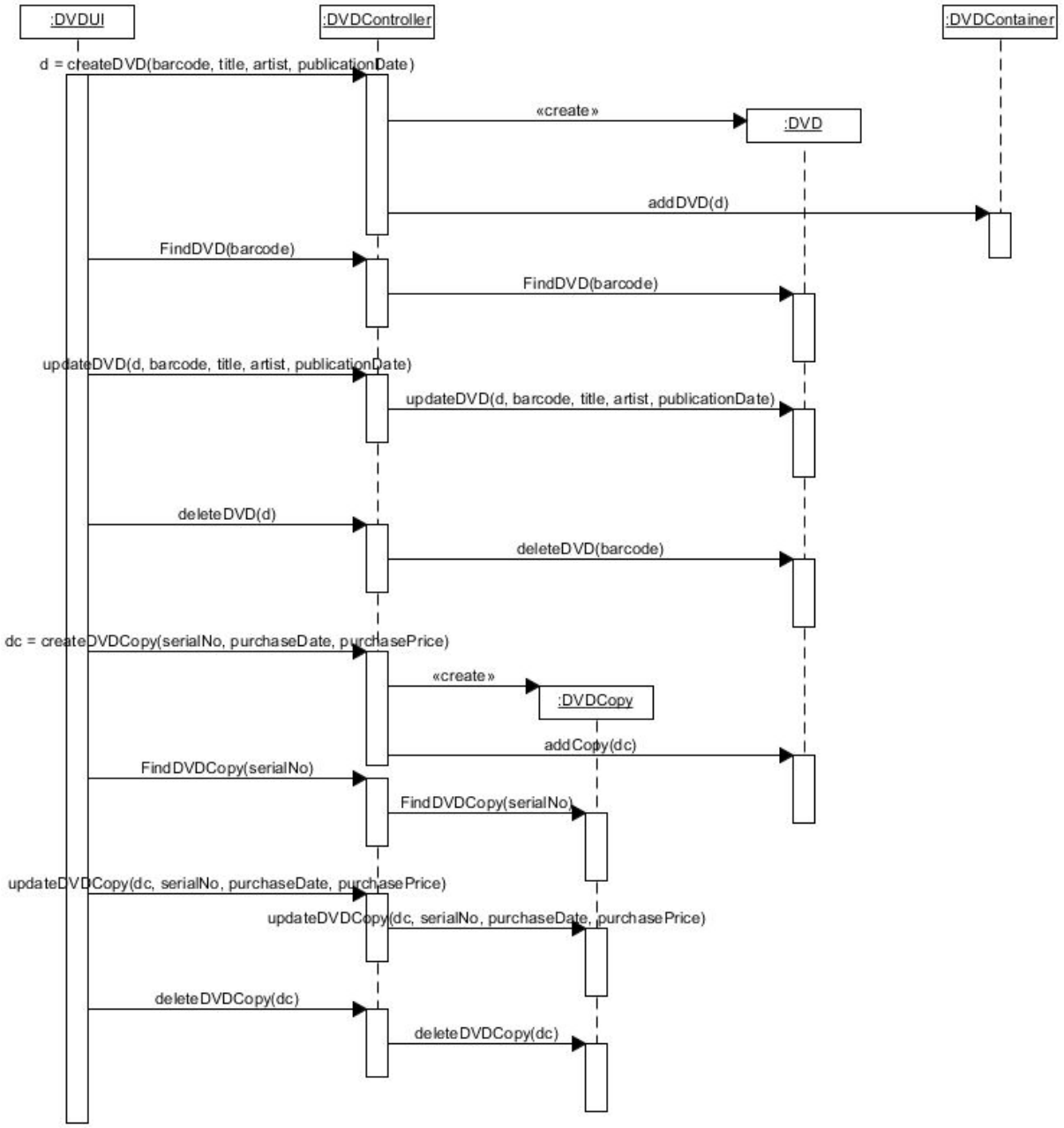
Use case navn	Manage Copy - CRUD	
Aktør	Joe	
Prebetingelser	DVD er i systemet	
Postbetingelser	DVD opdateret	
Frekvens	0 -5 gange	
Flow of Events	**Aktørhandling**	**System svar**
	1. Joe åbner MainMenuUI	
	2. Joe opdaterer DVDCopy ved at søge efter dens serialno.	3. Systemet finder DVDCopy
	4. Joe indtaster ændringer	5. systemet tilføjer ændringerne til DVDCopy'en

Use case navn	Manage Copy - CRUD	
Aktør	Joe	
Prebetingelser	DVD er i systemet	
Postbetingelser	DVD slettet	
Frekvens	0 -5 gange	
Flow of Events	**Aktørhandling**	**System svar**
	1. Joe åbner MainMenuUI	
	2. Joe sletter DVDCopy ved at søge efter dens serialno.	3. Systemet finder DVDCopy og sletter DVDCopy

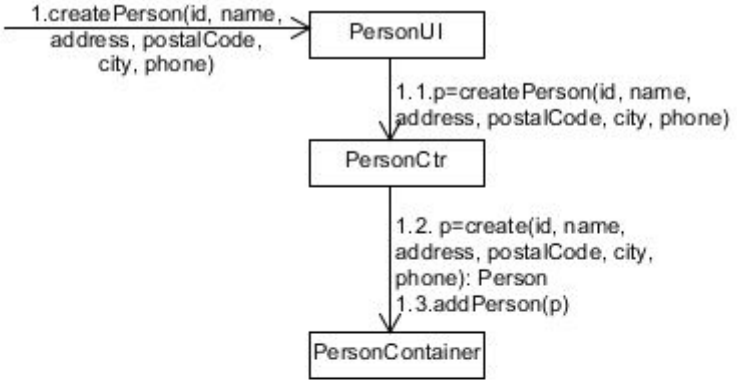
1.8

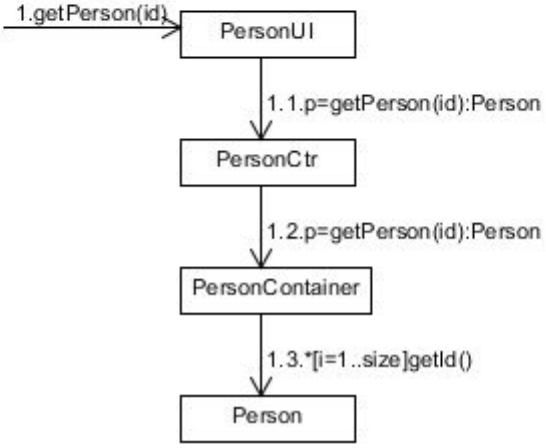

:LoanUI
:Person
:DVD
:DVDCopy
:LoanController
:LoanContainer
:Loan
getDVD(barcode)
getPerson(id)
getDVDCopy(serialNo)
createLoan(id, copies, borrowDate, period, status)
updateLoan(id, borrowDate, period, status)
FindLoan(id)
deleteLoan(loan)
addLoan(loan)
«find»
«create»
FindLoan(id)
deleteLoan(id)

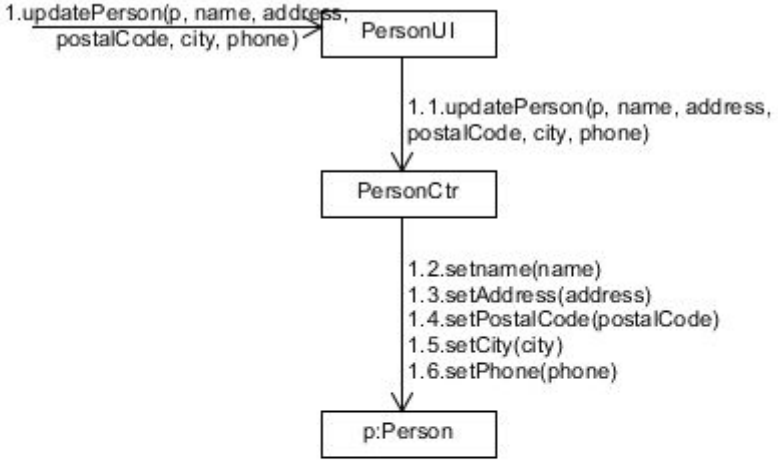
1.9

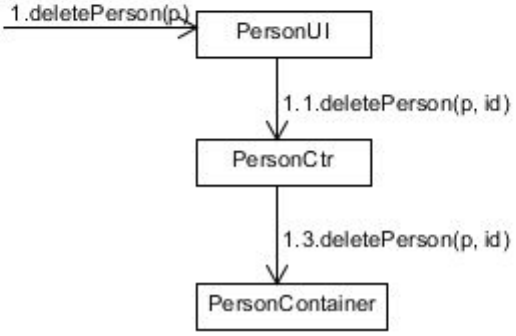

:DVDUI
:DVDController
:DVDContainer
d = createDVD(barcode, title, artist, publicationDate)
«create»
:DVD
addDVD(d)
FindDVD(barcode)
FindDVD(barcode)
updateDVD(d, barcode, title, artist, publicationDate)
updateDVD(d, barcode, title, artist, publicationDate)
deleteDVD(d)
deleteDVD(barcode)
dc = createDVDCopy(serialNo, purchaseDate, purchasePrice)
«create»
:DVDCopy
addCopy(dc)
FindDVDCopy(serialNo)
FindDVDCopy(serialNo)
updateDVDCopy(dc, serialNo, purchaseDate, purchasePrice)
updateDVDCopy(dc, serialNo, purchaseDate, purchasePrice)
deleteDVDCopy(dc)
deleteDVDCopy(dc)

2.0

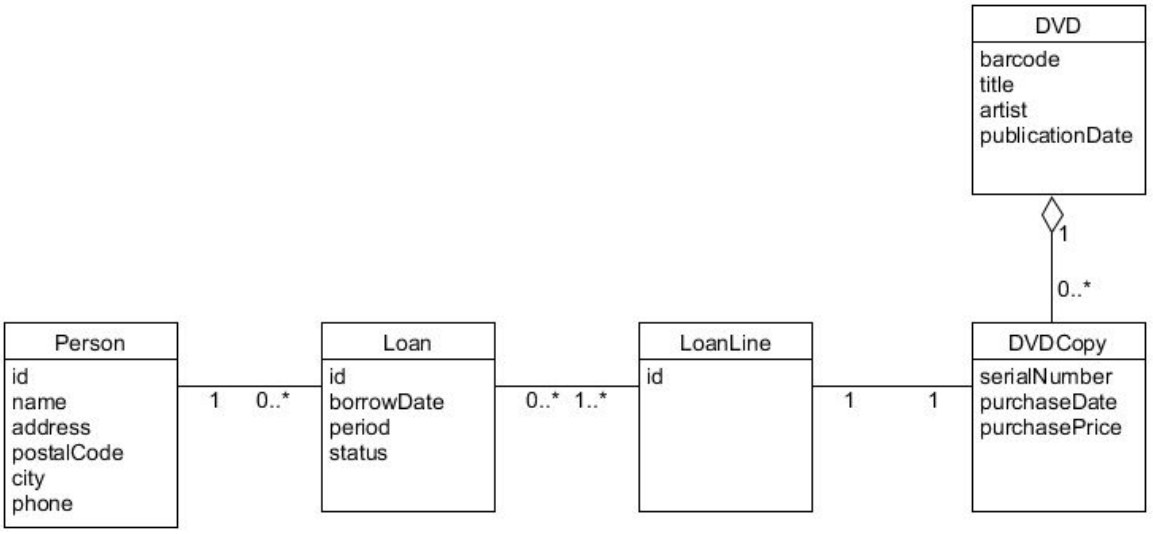

1.createPerson(id, name, address, postalCode, city, phone)
PersonUI
1.1.p=createPerson(id, name, address, postalCode, city, phone)
PersonCtr
1.2. p=create(id, name, address, postalCode, city, phone): Person
1.3.addPerson(p)
PersonContainer


1.getPerson(id)
PersonUI
1.1.p=getPerson(id):Person
PersonCtr
1.2.p=getPerson(id):Person
PersonContainer
1.3.*[i=1..size]getId()
Person


1.updatePerson(p, name, address, postalCode, city, phone)
PersonUI
1.1.updatePerson(p, name, address, postalCode, city, phone)
PersonCtr
1.2.setname(name)
1.3.setAddress(address)
1.4.setPostalCode(postalCode)
1.5.setCity(city)
1.6.setPhone(phone)
p:Person


1.deletePerson(p)
PersonUI
1.1.deletePerson(p, id)
PersonCtr
1.3.deletePerson(p, id)
PersonContainer

2.1



2.2

Operation: updateDVD (*d*, barcode, title, artist, publicationDate)
Use case: Manage DVD – CRUD
Pre-betingelser: DVD-objekt *d* fundet ud fra barcode-match
Post-betingelser:

- d.barcode er blevet til barcode,
  d.title er blevet til title,
  d.artist er blevet til artist,
  d.publicationDate er blevet til publicationDate

2.3

Operation: deleteDVD (*d*)
Use case: Manage DVD – CRUD
Pre-betingelser: DVD-objekt *d* fundet ud fra barcode-match
Post-betingelser:

- *d* slettet

2.4

Operation: createCopy (serialNo, purchaseDate, purchasePrice)
Use case: Manage Copy – CRUD
Pre-betingelser: CopyContainer eksisterer
Post-betingelser:

- Et DVDCopy-objekt *dc* er blevet oprettet
- *dc* er associeret til CopyContainer
- *dc*.serialNo er blevet til serialNo,
  dc.purchaseDate er blevet til purchaseDate,
  dc.purchasePrice er blevet til purchasePrice

2.5

Operation: updateCopy (*dc*, serialNo, purchaseDate, purchasePrice)
Use case: Manage Copy – CRUD
Pre-betingelser: DVDCopy-objekt *dc* fundet ud fra serialNo-match
Post-betingelser:

- *dc*.serialNo er blevet til serialNo,
  dc.purchaseDate er blevet til purchaseDate,
  dc.purchasePrice er blevet til purchasePrice

2.6

Operation: deleteCopy (*dc*)
Use case: Manage Copy – CRUD
Pre-betingelser: DVDCopy-objekt *dc* fundet ud fra serialNo-match
Post-betingelser:

- *dc* slettet

2.7

Operation: lendDVD (*p*, *d*, *dc*)
Use case: Lend DVD
Pre-betingelser: LoanContainer eksisterer, *p* objekt, *d* objekt, *dc* objekt er oprettet
Post-betingelser:

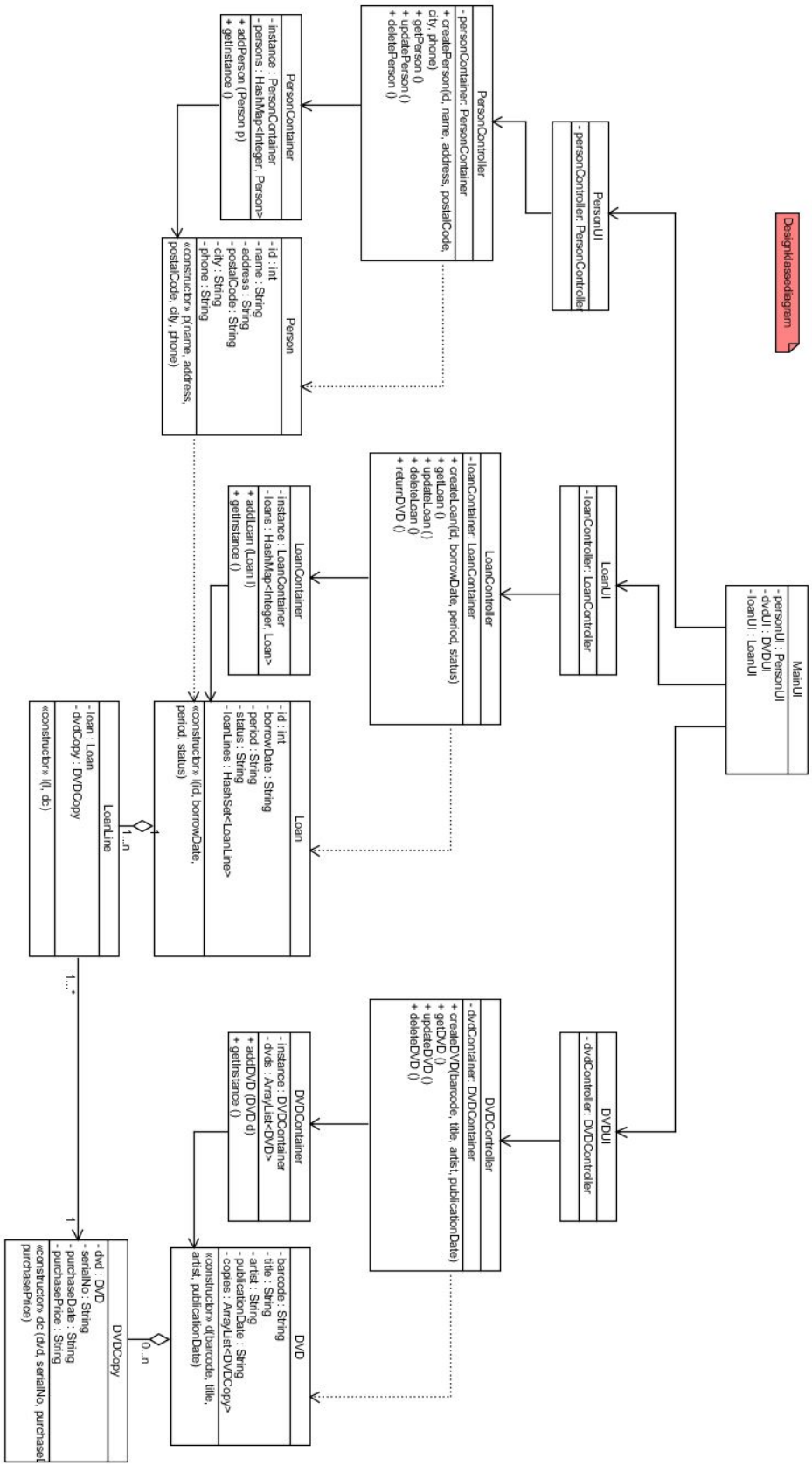
- Et Loan-objekt 'loan' er blevet oprettet
- *loan* er associeret til *LoanContainer*
- *p* er associeret til loan
- *d* er associeret til loan
- *dc* er associeret til loan
- loan.id er blevet til id,
  loan.borrowDate er blevet til borrowDate,
  loan.period er blevet til period,
  loan.status er blevet til active

2.8

Operation: returnDVD (*loan*, id)
Use case: Return DVD
Pre-betingelser: loan objekt eksisterer og er fundet ud fra id-match
Post-betingelser:

- Loan.status er blevet til inactive

2.9


Designklassediagram
MainUI
- personUI : PersonUI
- dvdUI : DVDUI
- loanUI : LoanUI
PersonUI
- personController: PersonController
PersonController
- personContainer: PersonContainer
+ createPerson(id, name, address, postalCode, city, phone)
+ getPerson ()
+ updatePerson ()
+ deletePerson ()
PersonContainer
- instance : PersonContainer
- persons : HashMap<Integer, Person>
+ addPerson (Person p)
+ getInstance ()
Person
- id : int
- name : String
- address : String
- postalCode : String
- city : String
- phone : String
«constructor» p(name, address, postalCode, city, phone)
LoanUI
- loanController: LoanController
LoanController
- loanContainer: LoanContainer
+ createLoan(id, borrowDate, period, status)
+ getLoan ()
+ updateLoan ()
+ deleteLoan ()
+ returnDVD ()
LoanContainer
- instance : LoanContainer
- loans : HashMap<Integer, Loan>
+ addLoan (Loan l)
+ getInstance ()
Loan
- id : int
- borrowDate : String
- period : String
- status : String
- loanLines : HashSet<LoanLine>
«constructor» l(id, borrowDate, period, status)
LoanLine
- loan : Loan
- dvdCopy : DVDCopy
«constructor» ll(, dc)
1
1...n
1...*
DVDUI
- dvdController: DVDController
DVDController
- dvdContainer: DVDContainer
+ createDVD(barcode, title, artist, publicationDate)
+ getDVD ()
+ updateDVD ()
+ deleteDVD ()
DVDContainer
- instance : DVDContainer
- dvds : ArrayList<DVD>
+ addDVD (DVD d)
+ getInstance ()
DVD
- barcode : String
- title : String
- artist : String
- publicationDate : String
- copies : ArrayList<DVDCopy>
«constructor» d(barcode, title, artist, publicationDate)
DVDCopy
- dvd : DVD
- serialNo : String
- purchaseDate : String
- purchasePrice : String
«constructor» dc (dvd, serialNo, purchaseDate, purchasePrice)
1
0...n

3.1

A	B	C	D	E	F
	05/11	**06/11**	**07/11**	**08/11**	**09/11**
Kodekonvention					
Iteration 1					
Iteration 2					
Iteration 3					
Iteration 4					
Programmering					
Rapportskrivning					
Refleksion					
Færdiggjort					
Igangværende					
Overarbejde					

Tidsplan for arbejdsproces.

3.2 - Kode

MainMenuUI.java

```
package tui;
import controller.*;
import model.*;
import java.time.LocalDateTime;

public class MainMenuUI {
  public static void main(String args[]) {
      new MainMenuUI().start();
  }

  private void start() {
      printMenu();
      boolean run = true;

      while(run) {
          String command = Helper.readInput("What is your command?");
          switch(command.toLowerCase()) {
              case "exit":
              case "quit":
              case "q":
                  run = false;
                  break;
              case "help":
                  printMenu();
              case "person menu":
                  PersonUI.start();
                  break;
              case "dvd menu":
                  DVDUI.start();
                  break;
              case "loan menu":
                  LoanUI.start();
                  break;
              case "test me":
                  runTests();
                  break;
              default:
                  System.out.println("Command not recognized.. Try again!");
          }
      }
      System.out.println("System terminated...");
```

```
    }

    private void printMenu() {
        System.out.println("************************");
        System.out.println("*       Main Menu      *");
        System.out.println("*                      *");
        System.out.println("*   Available options:  *");
        System.out.println("*  Quit: q, quit, exit  *");
        System.out.println("*  Person menu: person  *");
        System.out.println("*     DVD menu: dvd     *");
        System.out.println("*    Loan menu: loan    *");
        System.out.println("************************");
    }

    private void runTests() {
        PersonController pCtrl = new PersonController();
        DVDController dCtrl = new DVDController();
        LoanController lCtrl = new LoanController();

        pCtrl.createPerson("Jimmy", "Langagervej 4, st. 69", "9220", "Aalborg Øst",
"31337135");
        pCtrl.createPerson("Kongen", "Kongevej 4", "2400", "København", "1337");

        dCtrl.createDVD("123456", "Saw", "James Cameron", "07/11-2007");
        dCtrl.createDVD("654321", "Saw IV", "James Cameron", "05/12-2008");
        DVD d = dCtrl.getDVD("123456");
        DVD d1 = dCtrl.getDVD("654321");

        d.addCopy(new DVDCopy("asd", "22/12-1221", "20$", d));
        d.addCopy(new DVDCopy("asdasd", "25/12-1221", "20$", d));
        d1.addCopy(new DVDCopy("asdasdasd", "25/12-1221", "20$", d1));
        d1.addCopy(new DVDCopy("asdasdasdasd", "25/12-1221", "20$", d1));

        // create a loan
        Person p = pCtrl.getPerson(1);
        LocalDateTime borrowDate =
LocalDateTime.from(Helper.currentDateTime().minusDays(1));
        Loan l = lCtrl.createLoan(p, borrowDate, "30");
        LoanLine ll = new LoanLine(l, d.getDVDCopy("asd"));
        l.addLoanLine(ll);
    }
}
```

PersonUI.java

```
package tui;
import controller.*;
import model.*;

public final class PersonUI {
 public static void start() {
  printMenu();
  boolean run = true;

  while(run) {
    String command = Helper.readInput("What is your command?");
    switch (command.toLowerCase()) {
      case "help":
        printMenu();
        break;
      case "back":
        System.out.println("Going back to main menu ...");
        run = false;
        break;
      case "add person":
        addPerson();
        break;
      case "find person":
        findPerson();
        break;
      case "update person":
        updatePerson();
        break;
      case "delete person":
        deletePerson();
        break;
    }
  }
}

private static void printMenu() {
  System.out.println("**********************************");
  System.out.println("*          Person Menu           *");
  System.out.println("*                                *");
  System.out.println("*       Available options:       *");
  System.out.println("*        Add person: back        *");
  System.out.println("*    Find person: find person    *");
  System.out.println("*  Update person: update person  *");
  System.out.println("*  Delete person: delete person  *");
```

```
    System.out.println("********************************");
}

private static void addPerson() {
    PersonController pCtrl = new PersonController();

    String name = Helper.readInput("Enter name");
    String address = Helper.readInput("Enter address");
    String postalCode = Helper.readInput("Enter postal code");
    String city = Helper.readInput("Enter city");
    String phone = Helper.readInput("Enter phone number");

    if(Helper.checkEmptyInput(name)
      || Helper.checkEmptyInput(address)
      || Helper.checkEmptyInput(postalCode)
      || Helper.checkEmptyInput(city)
      || Helper.checkEmptyInput(phone)) {
        System.out.println("You missed some information ... Try again!");
    } else {
      Person p = pCtrl.createPerson(name, address, postalCode, city, phone);
      System.out.println("Person added!");
    }
}

private static void findPerson() {
    PersonController pCtrl = new PersonController();

    int id = Helper.readInt("Enter the person's id");
    Person p = pCtrl.getPerson(id);

    if(p != null) {
      System.out.println(p);
      if(pCtrl.getLoans(p).size() > 0) {
        System.out.println(p.getName() + " has " + pCtrl.getLoans(p).size() + " loans:");
        for(Loan l : pCtrl.getLoans(p).values()) {
          System.out.println(l);
        }
      } else
        System.out.println("Person has no loans!");
    } else
      System.out.println("Person not found!");
}

private static void updatePerson() {
    PersonController pCtrl = new PersonController();
    String res;
```

```
  int id = Helper.readInt("Enter the person's id");
  Person p = pCtrl.getPerson(id);

  if(p != null) {
    String name = Helper.readInput("Enter new name");
    String address = Helper.readInput("Enter new address");
    String postalCode = Helper.readInput("Enter new postal code");
    String city = Helper.readInput("Enter new city");
    String phone = Helper.readInput("Enter new phone number");

    if(!Helper.checkEmptyInput(name))
      p.setName(name);
    if(!Helper.checkEmptyInput(address))
      p.setAddress(address);
    if(!Helper.checkEmptyInput(postalCode))
      p.setPostalCode(postalCode);
    if(!Helper.checkEmptyInput(city))
      p.setCity(city);
    if(!Helper.checkEmptyInput(phone))
      p.setPhone(phone);
    res = "The person was updated!";
  } else
    res = "Person was not found!";

  System.out.println(res);
}

private static void deletePerson() {
  PersonController pCtrl = new PersonController();

  int id = Helper.readInt("Enter the person's id");
  Person p = pCtrl.getPerson(id);

  pCtrl.deletePerson(p);

  System.out.println(p.getName() + " was deleted!");
}
}
```

DVDUI.java

```
package tui;
import controller.*;
import model.*;

public final class DVDUI {
 public static void start() {
  printMenu();
  boolean run = true;

  while(run) {
    String command = Helper.readInput("What is your command?");
    switch (command.toLowerCase()) {
      case "help":
        printMenu();
        break;
      case "back":
        System.out.println("Going back to main menu ...");
        run = false;
        break;
      case "add dvd":
        addDVD();
        break;
      case "find dvd":
        findDVD();
        break;
      case "update dvd":
        updateDVD();
        break;
      case "delete dvd":
        deleteDVD();
        break;
      case "add dvd copy":
        addDVDCopy();
        break;
      case "delete dvd copy":
        deleteDVDCopy();
        break;
    }
  }
}

 private static void printMenu() {
  System.out.println("**************************************");
  System.out.println("*              DVD Menu              *");
```

```
  System.out.println("*                                  *");
  System.out.println("*          Available options:      *");
  System.out.println("*           Main menu: back        *");
  System.out.println("*           Add DVD: add dvd       *");
  System.out.println("*          Find DVD: find dvd      *");
  System.out.println("*        Update DVD: update dvd    *");
  System.out.println("*        Delete DVD: delete dvd    *");
  System.out.println("*     Add DVD Copy: add dvd copy   *");
  System.out.println("*  Delete DVD Copy: delete dvd copy *");
  System.out.println("************************************");
}

private static void addDVD() {
  DVDController dvdCtrl = new DVDController();
  String barcode = Helper.readInput("Enter the DVD's barcode");
  String title = Helper.readInput("Enter the DVD's title");
  String artist = Helper.readInput("Enter the DVD's artist");
  String pubDate = Helper.readInput("Enter the DVD's privateation date");

  DVD d = dvdCtrl.createDVD(barcode, title, artist, pubDate);
  if(d != null) {
    System.out.println("A new DVD was created!");
  }
  else
    System.out.println("An error occurred ...");
}

private static void findDVD() {
  DVDController dvdCtrl = new DVDController();
  String barcode = Helper.readInput("Enter the DVD's barcode");

  DVD d = dvdCtrl.getDVD(barcode);
  if(d != null)
    System.out.println(d);
  else
    System.out.println("DVD not found!");
}

private static void updateDVD() {
  DVDController dvdCtrl = new DVDController();
  String res;
  String inputBarcode = Helper.readInput("Enter the DVD's barcode");

  DVD d = dvdCtrl.getDVD(inputBarcode);

  if(d != null) {
```

```
    String barcode = Helper.readInput("Enter new barcode");
    String title = Helper.readInput("Enter new title");
    String artist = Helper.readInput("Enter new artist");
    String pubDate = Helper.readInput("Enter new privateation date");

    dvdCtrl.updateDVD(d, barcode, title, artist, pubDate);
    res = "The DVD was updated!";
  } else
    res = "DVD was not found!";

  System.out.println(res);
}

private static void deleteDVD() {
  DVDController dvdCtrl = new DVDController();

  String inputBarcode = Helper.readInput("Enter the DVD's barcode");
  DVD d = dvdCtrl.getDVD(inputBarcode);

  dvdCtrl.deleteDVD(d);

  System.out.println(d.getTitle() + " was deleted!");
}

private static void addDVDCopy() {
  DVDController dvdCtrl = new DVDController();

  String inputBarcode = Helper.readInput("Enter the DVD's barcode");
  String serialNo = Helper.readInput("Enter the DVD copy's serial no.");
  String purchaseDate = Helper.readInput("Enter the DVD copy's purchase date");
  String purchasePrice = Helper.readInput("Enter the DVD copy's purchase price");

  DVD d = dvdCtrl.getDVD(inputBarcode);
  DVDCopy dc = new DVDCopy(serialNo, purchaseDate, purchasePrice, d);

  if(d.addCopy(dc))
    System.out.println("DVD copy added!");
  else
    System.out.println("An error occurred ...");
}

private static void deleteDVDCopy() {
  DVDController dvdCtrl = new DVDController();

  String inputBarcode = Helper.readInput("Enter the DVD's barcode");
  String inputSerialNo = Helper.readInput("Enter the DVD copy's serial no.");
```

```
  DVD d = dvdCtrl.getDVD(inputBarcode);
  DVDCopy dc = d.getDVDCopy(inputSerialNo);

  if(dc != null) {
    if (d.removeCopy(dc))
      System.out.println("DVD copy with serial no. " + dc.getSerialNo() + " was deleted!");
  } else
    System.out.println("DVD copy was not found ...");
}
}
```

LoanUI.java

```
package tui;
import controller.*;
import model.*;

public final class LoanUI {
 public static void start() {
  printMenu();
  boolean run = true;

  while(run) {
   String command = Helper.readInput("What is your command?");
   switch (command.toLowerCase()) {
     case "help":
       printMenu();
       break;
     case "back":
       System.out.println("Going back to main menu ...");
       run = false;
       break;
     case "add loan":
       addLoan();
       break;
     case "find loan":
       findLoan();
       break;
     case "return loan":
       returnLoan();
       break;
     case "reopen loan":
       reOpenLoan();
       break;
     case "all loans":
       allLoans();
       break;
     case "all active loans":
       allActiveLoans();
       break;
     case "all returned loans":
       allInActiveLoans();
       break;
     case "return date":
       getReturnDate();
       break;
     default:
```

```
        System.out.println("Command not recognized.. Try again!");
        break;
    }
  }
}

private static void printMenu() {
  System.out.println("*****************************************");
  System.out.println("*                Loan Menu              *");
  System.out.println("*                                       *");
  System.out.println("*            Available options:         *");
  System.out.println("*             Main menu: back           *");
  System.out.println("*            Add loan: add loan         *");
  System.out.println("*           Find loan: find loan        *");
  System.out.println("*         Return date: return date      *");
  System.out.println("*         Return loan: return loan      *");
  System.out.println("*         Re-open loan: reopen loan     *");
  System.out.println("*            All loans: all loans       *");
  System.out.println("*    All active loans: all active loans *");
  System.out.println("* All in-active loans: all returned loans *");
  System.out.println("*****************************************");
}

private static void addLoan() {
  LoanController lCtrl = new LoanController();
  PersonController pCtrl = new PersonController();
  DVDController dCtrl = new DVDController();
  boolean run = true;
  String input;

  // find person
  int pId = Helper.readInt("Enter the person's ID");
  Person p = pCtrl.getPerson(pId);

  // create a loan
  Loan l = lCtrl.createLoan(p, Helper.currentDateTime(), "30");

  System.out.println("Begin adding dvd's to your loan!");
  System.out.println("Type :done when you're done.");
  while(run) {
    String dBarcode = Helper.readInput("Enter the DVD's barcode");
    switch(dBarcode) {
      case ":done":
        if(l.getLoanLines().size() == 0) {
          lCtrl.deleteLoan(l);
          System.out.println("Loan was not added because you did not add any loan lines!");
```

```
        } else
          System.out.println("A loan was added!");
        run = false;
        break;
      default:
        try {
          // find dvd
          DVD d = DVDContainer.getInstance().findDVD(dBarcode);

          // find dvd copy
          String dcSerialNo = Helper.readInput("Enter the DVD Copy's serial no.");
          DVDCopy dc = d.getDVDCopy(dcSerialNo);

          // add loan line
          if (d.getTitle() != null && dc.getSerialNo() != null) {
            LoanLine ll = new LoanLine(l, dc);

            // if addLoanLine returns false then the dvd copy is already rented
            if(!l.addLoanLine(ll))
              System.out.println("That DVD copy is already rented!");
          }
        } catch (NullPointerException e) {
          System.out.println("DVD or DVD copy could not be found!");
        }
        break;
    }
  }
}

private static void findLoan() {
  LoanController lCtrl = new LoanController();

  int lId = Helper.readInt("Enter the loan's ID");
  Loan l = lCtrl.getLoan(lId);

  if(l != null)
    System.out.println(l);
  else
    System.out.println("Loan could not be found!");
}

private static void returnLoan() {
  LoanController lCtrl = new LoanController();

  int lId = Helper.readInt("Enter the loan's ID");
  Loan l = lCtrl.getLoan(lId);
```

```
  if(Helper.currentDateTime().isBefore(l.getBorrowDate())) {
    lCtrl.deactivateLoan(l);
    System.out.println("Loan is now returned!");
  } else {
    l.setStatus("overdue");
    System.out.println("That loan is overdue .. Idiot!");
  }


}

private static void reOpenLoan() {
  LoanController lCtrl = new LoanController();

  int lId = Helper.readInt("Enter the loan's ID");
  Loan l = lCtrl.getLoan(lId);
  lCtrl.activateLoan(l);
  l.setBorrowDate(Helper.currentDateTime());

  System.out.println("Loan is now active!");
}

private static void allLoans() {
  for(Loan l : LoanContainer.getInstance().findAll().values()) {
    System.out.println(l);
  }
}

private static void allActiveLoans() {
  System.out.println("Listing active loans ...");
  for(Loan l : LoanContainer.getInstance().findAllActive().values()) {
    System.out.println(l);
  }
}

private static void allInActiveLoans() {
  System.out.println("Listing returned loans ...");
  for(Loan l : LoanContainer.getInstance().findAllInActive().values()) {
    System.out.println(l);
  }
}

private static void getReturnDate() {
  int lId = Helper.readInt("Enter loan ID");
  Loan l = LoanContainer.getInstance().findLoan(lId);
```

```
    System.out.println(Helper.formatDateTime(Helper.returnDateTime(l.getBorrowDate(),
l.getPeriod())));
  }
}
```

Helper.java

```
package tui;

import java.time.LocalDate;
import java.util.Scanner;
import java.time.format.DateTimeFormatter;
import java.time.LocalDateTime;

public final class Helper {
 private Helper() {
  throw new UnsupportedOperationException();
 }

 public static String readInput(String question) {
  System.out.print(question + ": ");
  Scanner s = new Scanner(System.in);
  return s.nextLine();
 }

 public static int readInt(String question) {
  System.out.print(question + ": ");
  Scanner s = new Scanner(System.in);
  return s.nextInt();
 }

 public static boolean checkEmptyInput(String input) {
  boolean res = false;
  if(input.equals("") || input.equals(" "))
   res = true;
  return res;
 }

 public static LocalDateTime currentDateTime() {
  return LocalDateTime.now();
 }

 public static String formatDateTime(LocalDateTime dateTime) {
  DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd/MM-yyyy HH:mm:ss");
  return dtf.format(dateTime);
 }

 public static LocalDateTime returnDateTime(LocalDateTime b, String p) {
  int period = Integer.parseInt(p);
  return LocalDateTime.from(b.plusDays(period));
 }
```

DVDContainer.java

```
package model;
import java.util.*;

public class DVDContainer {
   private static DVDContainer instance = null;
   private ArrayList<DVD> dvds;

   private DVDContainer() {
      dvds = new ArrayList<>();
   }

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

   public boolean addDVD(DVD d) {
      return dvds.add(d);
   }

   public DVD findDVD(String barcode) {
      DVD current = null;
      DVD res = null;
      int index = 0;
      boolean found = false;
      int size = dvds.size();
      while (index < size && !found) {
         current = dvds.get(index);
         if(current.getBarcode().equals(barcode)) {
            found = true;
         } else {
            index++;
         }
      }
      if (found) {
         res = dvds.get(index);
      }
      return res;
   }

   public DVD deleteDVD(String barcode) {
      DVD current = null;
```

```
        DVD res = null;
        int index = 0;
        boolean found = false;
        int size = dvds.size();
        while (index < size && !found) {
            current = dvds.get(index);
            if(current.getBarcode().equals(barcode)) {
                found = true;
            } else {
                index++;
            }
        }
        if(found) {
            res = dvds.get(index);
            dvds.remove(index);
        }
        return res;
    }
}
```

PersonContainer.java

```
package model;
import java.util.*;

public class PersonContainer {
   private static PersonContainer instance = null;
   private HashMap<Integer, Person> persons;

   private PersonContainer(){
      persons = new HashMap<Integer, Person>();
   }

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

   public boolean addPerson(Person p) {
      if(persons.put(p.getId(), p) != null)
         return true;
      return false;
   }

   public Person findPerson(int id) {
      return persons.get(id);
   }

   public HashMap<Integer, Person> findAll() {
      HashMap<Integer, Person> res = new HashMap<Integer, Person>(persons);
      return res;
   }

   public Person deletePerson(int id){
      Person p = findPerson(id);
      persons.remove(p.getId());
      return p;
   }

   // used to determine which id to give a new person upon creation
   public int getLastPersonId() {
      int res = 0;
      Person p;
      if(!persons.isEmpty()) {
```

```
            p = persons.get(persons.size());
            res = p.getId();
        }
        return res;
    }

    public HashMap<Integer, Loan> findAllLoans(Person p) {
        HashMap<Integer, Loan> loans = new HashMap<Integer,
Loan>(LoanContainer.getInstance().findAll());
        HashMap<Integer, Loan> res = new HashMap<Integer, Loan>();

        Loan current = null;
        int size = loans.size();
        for(int i = 1; i <= size; i++) {
            current = loans.get(i);
            if(current.getPerson().getId() == p.getId()) {
                res.put(current.getId(), current);
            }
        }
        return res;
    }
}
```

LoanContainer.java

```
package model;
import tui.Helper;

import java.util.*;

public class LoanContainer {
 private static LoanContainer instance = null;
 private HashMap<Integer, Loan> loans;

 private LoanContainer() {
   loans = new HashMap<Integer, Loan>();
 }

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

 public boolean addLoan(Loan l) {
   loans.put(l.getId(), l);
   return true;
 }

 public Loan findLoan(int id) {
   return loans.get(id);
 }

 public HashMap<Integer, Loan> findAll() {
   HashMap<Integer, Loan> res = new HashMap<Integer, Loan>(loans);
   return res;
 }

 public HashMap<Integer, Loan> findAllActive() {
   HashMap<Integer, Loan> res = new HashMap<Integer, Loan>();

   for(int i = 1; i <= loans.size(); i++) {
     Loan current = loans.get(i);
     if(current.getStatus().equals("active")) {
       res.put(current.getId(), current);
     }
   }

   return res;
```

```
}

public HashMap<Integer, Loan> findAllInActive() {
  HashMap<Integer, Loan> res = new HashMap<Integer, Loan>();

  for(int i = 1; i <= loans.size(); i++) {
    Loan current = loans.get(i);
    if(current.getStatus().equals("inactive")) {
      res.put(current.getId(), current);
    }
  }

  return res;
}

public Loan deleteLoan(int id){
  Loan l = findLoan(id);
  loans.remove(l.getId());
  return l;
}

public boolean activateLoan(Loan l) {
  l.setStatus("active");
  l.setBorrowDate(Helper.currentDateTime());
  return true;
}

public boolean deactivateLoan(Loan l) {

  l.setStatus("inactive");
  return true;
}

public int getLastLoanId() {
  int res = 0;
  Loan l;
  if(!loans.isEmpty()) {
    l = loans.get(loans.size());
    res = l.getId();
  }
  return res;
}
}
```

PersonController.java

```
package controller;
import model.*;
import tui.Helper;
import java.util.HashMap;

public class PersonController  {
   public Person createPerson(String name, String address, String postalCode, String city,
String phone)  {
      int id = PersonContainer.getInstance().getLastPersonId() + 1;
      Person p = new Person(id, name, address, postalCode, city, phone);

      if (!PersonContainer.getInstance().addPerson(p)){
         p = null;
      }
      return p;
   }

   //getPerson
   public Person getPerson(int id){
      return PersonContainer.getInstance().findPerson(id);
   }

   //updatePerson
   public Person updatePerson(Person p, String name, String address, String postalCode,
String city, String phone) {
      if(!Helper.checkEmptyInput(name))
         p.setName(name);
      if(!Helper.checkEmptyInput(address))
         p.setAddress(address);
      if(!Helper.checkEmptyInput(postalCode))
         p.setPostalCode(postalCode);
      if(!Helper.checkEmptyInput(city))
         p.setCity(city);
      if(!Helper.checkEmptyInput(phone))
         p.setPhone(phone);
      return p;
   }

   //deletePerson
   public Person deletePerson(Person p) {
      return PersonContainer.getInstance().deletePerson(p.getId());
   }

   public HashMap<Integer, Loan> getLoans(Person p) {
```

```
        return PersonContainer.getInstance().findAllLoans(p);
    }
}
```

DVDController.java

```
package controller;
import model.*;
import tui.Helper;

public class DVDController {
    public DVD createDVD(String barcode, String title, String artist, String publicationDate) {
        DVD d = new DVD(barcode, title, artist, publicationDate);

        if(!DVDContainer.getInstance().addDVD(d))
            d = null;

        return d;
    }

    public DVD getDVD(String barcode) {
        return DVDContainer.getInstance().findDVD(barcode);
    }

    public DVD updateDVD(DVD d, String barcode, String title, String artist, String
publicationDate) {
        if(!Helper.checkEmptyInput(barcode))
            d.setBarcode(barcode);
        if(!Helper.checkEmptyInput(title))
            d.setTitle(title);
        if(!Helper.checkEmptyInput(artist))
            d.setArtist(artist);
        if(!Helper.checkEmptyInput(publicationDate))
            d.setPublicationDate(publicationDate);
        return d;
    }

    public DVD deleteDVD(DVD d) {
        return DVDContainer.getInstance().deleteDVD(d.getBarcode());
    }
}
```

LoanController.java

```
package controller;
import model.*;
import tui.Helper;
import java.time.LocalDateTime;

public class LoanController   {
 public Loan createLoan(Person p, LocalDateTime borrowDate, String period)   {
   int id = LoanContainer.getInstance().getLastLoanId() + 1;
   Loan l = new Loan(p, id, borrowDate, period);

   if (!LoanContainer.getInstance().addLoan(l)) {
     l = null;
   }
   return l;
}

// get loan
public Loan getLoan(int id){
   return LoanContainer.getInstance().findLoan(id);
}

public Loan deleteLoan(Loan l) {
   return LoanContainer.getInstance().deleteLoan(l.getId());
}

// change loan's status to active
public boolean activateLoan(Loan l) {
   LoanContainer.getInstance().activateLoan(l);
   return true;
}

// change loan's status to active
public boolean deactivateLoan(Loan l) {
   LoanContainer.getInstance().deactivateLoan(l);
   return true;
}
}
```

Person.java

```
package model;

public class Person {
   private int id;
   private String name;
   private String address;
   private String postalCode;
   private String city;
   private String phone;

   public Person (int id, String name, String address, String postalCode, String city, String
phone){
       this.id = id;
       this.name = name;
       this.address = address;
       this.postalCode = postalCode;
       this.city = city;
       this.phone = phone;
   }

   public void setId(int id) {
       this.id = id;
   }

   public int getId(){
      return id;
   }

   public void setName(String name){
       this.name = name;
   }

   public String getName(){
       return name;
   }

   public void setAddress (String address){
       this.address = address;
   }

   public String getAddress (){
       return address;
   }
```

```
  public void setPostalCode(String postalCode){
      this.postalCode = postalCode;
  }

  public String getPostalCode (){
      return postalCode;
  }

  public void setCity (String city){
      this.city = city;
  }

  public String getCity(){
      return city;
  }

  public void setPhone(String phone){
      this.phone = phone;
  }

  public String getPhone(){
      return phone;
  }

  public String toString() {
      String res = "Person:\n";
      res += "ID: " + this.getId() + "\n";
      res += "Name: " + this.getName() + "\n";
      res += "Address: " + this.getAddress() + "\n";
      res += "Postal code: " + this.getPostalCode() + "\n";
      res += "City: " + this.getCity() + "\n";
      res += "Phone: " + this.getPhone() + "\n";
      return res;
  }
}
```

PersonContainer.java

```
package model;
import java.util.*;

public class PersonContainer {
   private static PersonContainer instance = null;
   private HashMap<Integer, Person> persons;

   private PersonContainer(){
      persons = new HashMap<Integer, Person>();
   }

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

   public boolean addPerson(Person p) {
      if(persons.put(p.getId(), p) != null)
         return true;
      return false;
   }

   public Person findPerson(int id) {
      return persons.get(id);
   }

   public HashMap<Integer, Person> findAll() {
      HashMap<Integer, Person> res = new HashMap<Integer, Person>(persons);
      return res;
   }

   public Person deletePerson(int id){
      Person p = findPerson(id);
      persons.remove(p.getId());
      return p;
   }

   // used to determine which id to give a new person upon creation
   public int getLastPersonId() {
      int res = 0;
      Person p;
      if(!persons.isEmpty()) {
```

```
            p = persons.get(persons.size());
            res = p.getId();
        }
        return res;
    }

    public HashMap<Integer, Loan> findAllLoans(Person p) {
        HashMap<Integer, Loan> loans = new HashMap<Integer,
Loan>(LoanContainer.getInstance().findAll());
        HashMap<Integer, Loan> res = new HashMap<Integer, Loan>();

        Loan current = null;
        int size = loans.size();
        for(int i = 1; i <= size; i++) {
            current = loans.get(i);
            if(current.getPerson().getId() == p.getId()) {
                res.put(current.getId(), current);
            }
        }
        return res;
    }
}
```

DVD.java

```
package model;
import java.util.ArrayList;

public class DVD {
   private String barcode;
   private String title;
   private String artist;
   private String publicationDate;
   private ArrayList<DVDCopy> copies;

   public DVD(String barcode, String title, String artist, String publicationDate) {
      this.barcode = barcode;
      this.title = title;
      this.artist = artist;
      this.publicationDate = publicationDate;
      this.copies = new ArrayList<>();
   }

   public boolean addCopy(DVDCopy copy) {
      boolean res = true;
      if(copy != null)
         return copies.add(copy);
      else
         res = false;
      return res;
   }

   public DVDCopy getDVDCopy(String serialNo) {
      DVDCopy res = null;
      try {
         DVDCopy current =  null;
         int index = 0;
         boolean found = false;
         int size = copies.size();
         while (index < size && !found) {
            current = copies.get(index);
            if(current.getSerialNo().equals(serialNo)) {
               found = true;
            } else {
               index++;
            }
         }
         if (found) {
            res = copies.get(index);
```

```
        }
    } catch (NullPointerException e) {
        System.out.println("DVD could not be found ...");
    }
    return res;
}

public ArrayList<DVDCopy> getCopies() {
    ArrayList<DVDCopy> res = new ArrayList<DVDCopy>(copies);
    return res;
}

public boolean removeCopy(DVDCopy copy) {
    boolean res = true;
    if(copy != null)
        return copies.remove(copy);
    else
        res = false;
    return res;
}

public String getBarcode() {
    return barcode;
}

public void setBarcode(String barcode) {
    this.barcode = barcode;
}

public String getTitle() {
    return title;
}

public void setTitle(String title) {
    this.title = title;
}

public String getArtist() {
    return artist;
}

public void setArtist(String artist) {
    this.artist = artist;
}

public String getPublicationDate() {
```

```
        return publicationDate;
    }

    public void setPublicationDate(String publicationDate) {
        this.publicationDate = publicationDate;
    }

    public String toString() {
        String res = "DVD:\n";
        res += "Barcode: " + this.getBarcode() + "\n";
        res += "Title: " + this.getTitle() + "\n";
        res += "Artist: " + this.getArtist() + "\n";
        res += "Publication date: " + this.getPublicationDate() + "\n";
        if(copies.size() > 0) {
            res += "DVD Copies:\n";
            for(DVDCopy copy : copies) {
                res += copy;
                res += "\n";
            }
        } else
            res += "There are no copies of this DVD\n";

        return res;
    }
}
```

DVDContainer.java

```
package model;
import java.util.*;

public class DVDContainer {
   private static DVDContainer instance = null;
   private ArrayList<DVD> dvds;

   private DVDContainer() {
      dvds = new ArrayList<>();
   }

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

   public boolean addDVD(DVD d) {
      return dvds.add(d);
   }

   public DVD findDVD(String barcode) {
      DVD current = null;
      DVD res = null;
      int index = 0;
      boolean found = false;
      int size = dvds.size();
      while (index < size && !found) {
         current = dvds.get(index);
         if(current.getBarcode().equals(barcode)) {
            found = true;
         } else {
            index++;
         }
      }
      if (found) {
         res = dvds.get(index);
      }
      return res;
   }

   public DVD deleteDVD(String barcode) {
      DVD current = null;
```

```
        DVD res = null;
        int index = 0;
        boolean found = false;
        int size = dvds.size();
        while (index < size && !found) {
            current = dvds.get(index);
            if(current.getBarcode().equals(barcode)) {
                found = true;
            } else {
                index++;
            }
        }
        if(found) {
            res = dvds.get(index);
            dvds.remove(index);
        }
        return res;
    }
}
```

DVDCopy.java

```
package model;

public class DVDCopy {
    private String serialNo;
    private String purchaseDate;
    private String purchasePrice;
    private DVD dvd;

    public DVDCopy(String serialNo, String purchaseDate, String purchasePrice, DVD dvd) {
        this.serialNo = serialNo;
        this.purchaseDate = purchaseDate;
        this.purchasePrice = purchasePrice;
        this.dvd = dvd;
    }

    public String getSerialNo() {
        return serialNo;
    }

    public void setSerialNo(String serialNo) {
        this.serialNo = serialNo;
    }

    public String getPurchaseDate() {
        return purchaseDate;
    }

    public void setPurchaseDate(String purchaseDate) {
        this.purchaseDate = purchaseDate;
    }

    public String getPurchasePrice() {
        return purchasePrice;
    }

    public void setPurchasePrice(String purchasePrice) {
        this.purchasePrice = purchasePrice;
    }

    public DVD getDvd() {
        return dvd;
    }

    public void setDvd(DVD dvd) {
```

```
        this.dvd = dvd;
    }

    public String toString() {
        String res = "Serial no.: " + this.getSerialNo() + "\n";
        res += "Purchase date: " + this.getPurchaseDate() + "\n";
        res += "Purchase price: " + this.getPurchasePrice() + "\n";
        res += "DVD title: " + this.getDvd().getTitle() + "\n";
        return res;
    }
}
```

Loan.java

```
package model;
import tui.Helper;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.HashSet;
import java.util.HashMap;

public class Loan {
   private int id;
   private Person person;
   private LocalDateTime borrowDate;
   private String period;
   private String status; // active or inactive
   private HashSet<LoanLine> loanLines;

   public Loan(Person p, int id, LocalDateTime borrowDate, String period) {
      this.person = p;
      this.id = id;
      this.borrowDate = borrowDate;
      this.period = period;
      this.status = "active";
      this.loanLines = new HashSet<LoanLine>();
   }

   public boolean addLoanLine(LoanLine ll) {
      HashMap<Integer, Loan> activeLoans = new HashMap<Integer,
Loan>(LoanContainer.getInstance().findAllActive());
      DVDCopy dvdCopy = ll.getDVDCopy();
      boolean allowed = true;

      // check if the dvd copy is already rented
      for(Loan loan : activeLoans.values()) {
         for(LoanLine loanLine : loan.getLoanLines()) {
            if (loanLine.getDVDCopy().getSerialNo().equals(dvdCopy.getSerialNo()))
               allowed = false;
         }
      }

      // return true if the dvd copy is not rented and false if it is
      if(allowed && this.loanLines.add(ll))
         return true;
      return false;
   }
```

```
    public HashSet<LoanLine> getLoanLines() {
        return this.loanLines;
    }

    public void setPerson(Person p) {
        this.person = p;
    }

    public Person getPerson() {
        return person;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getId() {
        return id;
    }

    public void setBorrowDate(LocalDateTime borrowDate) {
        this.borrowDate = borrowDate;
    }

    public LocalDateTime getBorrowDate() {
        return borrowDate;
    }

    public String getBorrowDateString() { return Helper.formatDateTime(borrowDate); }

    public void setPeriod(String period) {
        this.period = period;
    }

    public String getPeriod() {
        return period;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public String getStatus() {
        return status;
    }
```

```
    @Override
    public String toString() {
        String res = "Loan:\n";
        res += "ID: " + this.getId() + "\t";
        res += "Loaner: " + this.getPerson().getName() + "\t";
        res += "Borrow date: " + this.getBorrowDateString() + "\t";
        res += "Period: " + this.getPeriod() + " days\t";
        res += "Status: " + this.getStatus() + "\n";
        res += "Loan lines:\n";
        for(LoanLine ll : loanLines) {
            res += ll;
        }
        return res;
    }
}
```

LoanContainer.java

```
package model;
import tui.Helper;

import java.util.*;

public class LoanContainer {
 private static LoanContainer instance = null;
 private HashMap<Integer, Loan> loans;

 private LoanContainer() {
   loans = new HashMap<Integer, Loan>();
 }

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

 public boolean addLoan(Loan l) {
   loans.put(l.getId(), l);
   return true;
 }

 public Loan findLoan(int id) {
   return loans.get(id);
 }

 public HashMap<Integer, Loan> findAll() {
   HashMap<Integer, Loan> res = new HashMap<Integer, Loan>(loans);
   return res;
 }

 public HashMap<Integer, Loan> findAllActive() {
   HashMap<Integer, Loan> res = new HashMap<Integer, Loan>();

   for(int i = 1; i <= loans.size(); i++) {
     Loan current = loans.get(i);
     if(current.getStatus().equals("active")) {
       res.put(current.getId(), current);
     }
   }

   return res;
```

```
}

public HashMap<Integer, Loan> findAllInActive() {
  HashMap<Integer, Loan> res = new HashMap<Integer, Loan>();

  for(int i = 1; i <= loans.size(); i++) {
    Loan current = loans.get(i);
    if(current.getStatus().equals("inactive")) {
      res.put(current.getId(), current);
    }
  }

  return res;
}

public Loan deleteLoan(int id){
  Loan l = findLoan(id);
  loans.remove(l.getId());
  return l;
}

public boolean activateLoan(Loan l) {
  l.setStatus("active");
  l.setBorrowDate(Helper.currentDateTime());
  return true;
}

public boolean deactivateLoan(Loan l) {

  l.setStatus("inactive");
  return true;
}

public int getLastLoanId() {
  int res = 0;
  Loan l;
  if(!loans.isEmpty()) {
    l = loans.get(loans.size());
    res = l.getId();
  }
  return res;
}
}
```

LoanLine.java

```
package model;

public class LoanLine {
 private Loan loan;
 private DVDCopy dvdCopy;

 public LoanLine(Loan l, DVDCopy dc) {
   this.loan = l;
   this.dvdCopy = dc;
 }

 public Loan getLoan() {
   return loan;
 }

 public void setLoan(Loan loan) {
   this.loan = loan;
 }

 public DVDCopy getDVDCopy() {
   return dvdCopy;
 }

 public void setDVDCopy(DVDCopy dc) {
   this.dvdCopy = dc;
 }

 @Override
 public String toString() {
   String res = "Title: " + this.getDVDCopy().getDvd().getTitle() + "\t";
   res += "Serial no.: " + this.getDVDCopy().getSerialNo() + "\n";
   return res;
 }
}
```

TestPerson.java

```
package model;

import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class TestPerson {
 private Person p;

 public TestPerson() {
 }

 @Before
 public void setUp() {
   p = new Person(1, "Peter Petersen", "Jensensvej 4", "9000", "Aalborg", "88888888");
 }

 @Test
 public void setId() {
   int newId = 99;

   p.setId(newId);

   assertEquals(99, p.getId());
 }

 @Test
 public void getId() {
   assertEquals(1, p.getId());
 }

 @Test
 public void setName() {
   String newName = "Peter Madsen";

   p.setName(newName);

   assertEquals(newName, p.getName());
 }

 @Test
 public void getName() {
   assertEquals("Peter Petersen", p.getName());
```

```
}

@Test
public void setAddress() {
  String newAddress = "Langagervej 4";

  p.setAddress(newAddress);

  assertEquals(newAddress, p.getAddress());
}

@Test
public void getAddress() {
  assertEquals("Jensensvej 4", p.getAddress());
}

@Test
public void setPostalCode() {
  String newPostalCode = "9220";

  p.setPostalCode(newPostalCode);

  assertEquals(newPostalCode, p.getPostalCode());
}

@Test
public void getPostalCode() {
  assertEquals("9000", p.getPostalCode());
}

@Test
public void setCity() {
  String newCity = "Aalborg Øst";

  p.setCity(newCity);

  assertEquals(newCity, p.getCity());
}

@Test
public void getCity() {
  assertEquals("Aalborg", p.getCity());
}

@Test
public void setPhone() {
```

```
    String newPhone = "33333333";

    p.setPhone(newPhone);

    assertEquals(newPhone, p.getPhone());
  }

  @Test
  public void getPhone() {
    assertEquals("88888888", p.getPhone());
  }
}
```

TestPersonContainer.java

```
package model;

import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class TestPersonContainer {
 private Person p;
 private Person p1;

 @Before
 public void setUp() {
   p = new Person(1, "Peter Petersen", "Jensensvej 4", "9000", "Aalborg", "88888888");
   p1 = new Person(2, "Jens Jensen", "Hejvej 4", "9000", "Aalborg", "33333333");
   PersonContainer.getInstance().addPerson(p);
   PersonContainer.getInstance().addPerson(p1);
 }

 @After
 public void tearDown(){
   PersonContainer.getInstance().deletePerson(p.getId());
   PersonContainer.getInstance().deletePerson(p1.getId());
   p = null;
   p1 = null;
 }

 @Test
 public void addPerson() {
   int expectedSizeOfPersonsList = 2;
   int actualSizeOfPersonsList = PersonContainer.getInstance().findAll().size();
   assertEquals(expectedSizeOfPersonsList, actualSizeOfPersonsList);
 }

 @Test
 public void findPerson() {
   Person expectedPerson = p;
   Person actualPerson = PersonContainer.getInstance().findPerson(p.getId());

   assertEquals(expectedPerson.getId(), actualPerson.getId());
   assertEquals(expectedPerson.getName(), actualPerson.getName());
 }

 @Test
```

```
public void deletePerson() {
  Person p2 = new Person(3, "Thomas Thomasen", "Hejvej 4", "9000", "Aalborg",
"33333333");
  PersonContainer.getInstance().addPerson(p2);

  PersonContainer.getInstance().deletePerson(p2.getId());
  int expectedSizeOfPersonsList = 2;
  int actualSizeOfPersonsList = PersonContainer.getInstance().findAll().size();
  assertEquals(expectedSizeOfPersonsList, actualSizeOfPersonsList);
}

@Test
public void getLastPersonId() {
  int lastId = PersonContainer.getInstance().getLastPersonId();

  assertEquals(2, lastId);
}
}
```

TestDVD.java

```
package model;

import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class TestDVD {
 private DVD d;

 @Before
 public void setUp() {
   d = new DVD("123456", "Saw IV", "Darren Lynn Bousman", "02/09-2007");
 }

 @After
 public void tearDown() {

 }

 @Test
 public void addCopy() {
   DVDCopy dc = new DVDCopy("9999999999", "05/01-2008", "20$", d);
   DVDCopy dc1 = new DVDCopy("9898989898", "05/01-2008", "20$", d);
   d.addCopy(dc);
   d.addCopy(dc1);
   int expectedCopies = 2;
   int actualCopies = d.getCopies().size();

   assertEquals(expectedCopies, actualCopies);
 }

 @Test
 public void getDVDCopy() {
   DVDCopy dc = new DVDCopy("9999999999", "05/01-2008", "20$", d);
   DVDCopy dc1 = new DVDCopy("9898989898", "08/01-2008", "20$", d);
   d.addCopy(dc);
   d.addCopy(dc1);

   String expectedCopyPurchaseDate = "05/01-2008";
   String actualCopyPurchaseDate = d.getDVDCopy(dc.getSerialNo()).getPurchaseDate();
   String expectedDVDTitle = "Saw IV";
   String actualDVDTitle = d.getDVDCopy(dc.getSerialNo()).getDvd().getTitle();
```

```
  assertEquals(expectedCopyPurchaseDate, actualCopyPurchaseDate);
  assertEquals(expectedDVDTitle, actualDVDTitle);
}

@Test
public void getCopies() {
  DVDCopy dc = new DVDCopy("9999999999", "05/01-2008", "20$", d);
  DVDCopy dc1 = new DVDCopy("9898989898", "08/01-2008", "20$", d);
  d.addCopy(dc);
  d.addCopy(dc1);

  int expectedCopies = 2;
  int actualCopies = d.getCopies().size();

  assertEquals(expectedCopies, actualCopies);
}

@Test
public void removeCopy() {
  DVDCopy dc = new DVDCopy("9999999999", "05/01-2008", "20$", d);
  DVDCopy dc1 = new DVDCopy("9898989898", "08/01-2008", "20$", d);
  d.addCopy(dc);
  d.addCopy(dc1);
  d.removeCopy(dc1);

  int expectedCopies = 1;
  int actualCopies = d.getCopies().size();

  assertEquals(expectedCopies, actualCopies);
}

@Test
public void getBarcode() {
  String expected = "123456";
  String actual = d.getBarcode();

  assertEquals(expected, actual);
}

@Test
public void setBarcode() {
  d.setBarcode("123");

  String expected = "123";
  String actual = d.getBarcode();
```

```
  assertEquals(expected, actual);
}

@Test
public void getTitle() {
  String expected = "Saw IV";
  String actual = d.getTitle();

  assertEquals(expected, actual);
}

@Test
public void setTitle() {
  d.setTitle("Saw VI");

  String expected = "Saw VI";
  String actual = d.getTitle();

  assertEquals(expected, actual);
}

@Test
public void getArtist() {
  String expected = "Darren Lynn Bousman";
  String actual = d.getArtist();

  assertEquals(expected, actual);
}

@Test
public void setArtist() {
  d.setArtist("James Cameron");

  String expected = "James Cameron";
  String actual = d.getArtist();

  assertEquals(expected, actual);
}

@Test
public void getPublicationDate() {
  String expected = "02/09-2007";
  String actual = d.getPublicationDate();

  assertEquals(expected, actual);
}
```

```
@Test
public void setPublicationDate() {
  d.setPublicationDate("05/09-2007");

  String expected = "05/09-2007";
  String actual = d.getPublicationDate();

  assertEquals(expected, actual);
}
}
```