## 3. Iteration 1

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

### 3.1 Use case

createPerson ”C”

**Primær aktør:** User

**Pre-betingelse:** En collection AddressBook er oprettet

**Post-betingelse:** Et Person objekt er oprettet og associeret med AddressBook

**Hoves succes scenarie:**

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

findPerson ”R”

**Primær aktør:** User

**Pre-betingelse:** Person objekt er oprettet

**Post-betingelse:** Person objekt er fundet

**Hoves succes scenarie:**

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

updatePerson ”U”

**Primær aktør:** User

**Pre-betingelse:** Person objekt er fundet

**Post-betingelse:** Systemet opdaterer person objekt

**Hoves succes scenarie:**

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

deletePerson ”D”

**Primær aktør:** User

**Pre-betingelse:** Person objekt er fundet i systemet

**Post-betingelse:** Systemet har slettet Person objekt

**Hoves succes scenarie:**

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

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

### 3.2 System Sekvensdiagram og operationskontrakter

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

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

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

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



Ovenstående SSD har følgende operationskontrakter.



### 3.3 Interaktionsdiagram

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

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

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

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

## Designklassediagrammet

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

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

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

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

