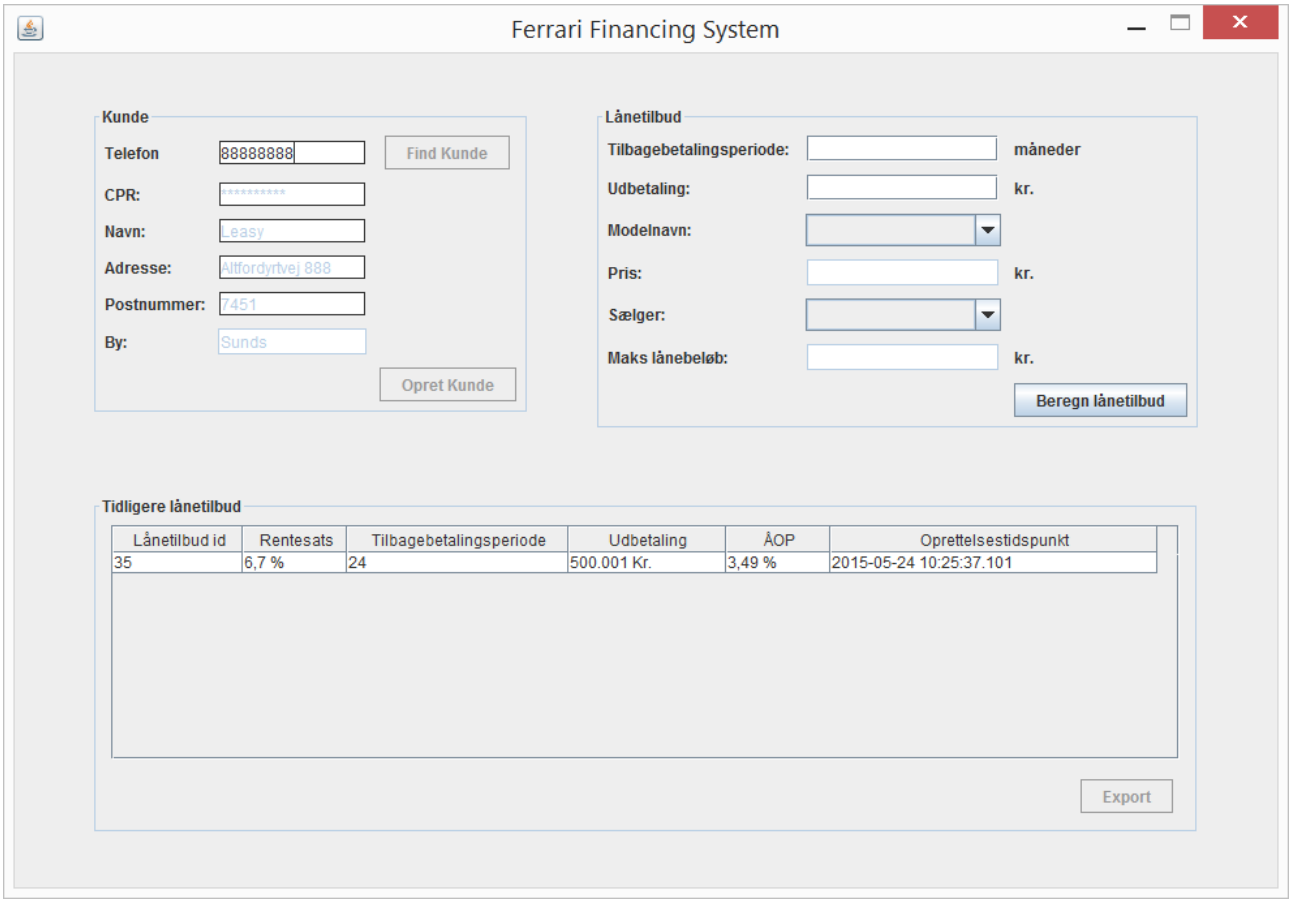


Anders Looft, Thomas Nielsen,

Simon Lorentsen og Lasse Meilby.

//he15dmu-2s14

// FFS

//Førsteårsprøven

//Erhvervsakademi MidtVest

//Datamatiker

//Vejleder: Anders W. Petersen, Flemming K. Jensen, Hans Iversen

//Juni 2015

//37.942 tegn

Erhvervsakademi MidtVest

Indholdsfortegnelse

Indledning (Thomas)

Vi har i dette projekt arbejdet iterativt over hver use case, og sågar igennem hele forløbet. Vi er gået ind i det med den indstilling, at vi laver dokumentation før implementation, og at vi ville nå så mange use cases som muligt. Opgaven lød på, at systemet skulle have et letforståelig og intuitivt interface, samt at feedback i brugergrænsefladen skulle være hurtig. Vi har designet og skabt en database, som kan indeholde oplysninger omkring kunder, sælgere, biler og aftaler. Det har været essentielt at cpr-numre skulle behandles med diskretion. Det var ønsket at en CSV-fil skulle kunne eksporteres, hvor en oversigt over lånetilbuddet, hvor en tilbagebetalingsplan indgår. Det har været hensigten at programmet skulle kunne videreføres til en webplatform, hvilket har afspejlet sig i nogle af de valg vi har truffet undervejs.

Design patterns

MVC pattern (Thomas)

Vi har i vores projekt valgt at adoptere en af de mest brugte design patterns, MVC-pattern, også kaldet model-view-controller. Dette har vi valgt og gøre, da vi allerede fra starten vidste, at dette design pattern ville passe fint til vores system. Dette grundes i at der er en masse kommunikation der skal behandles fra viewet og ned til modellen. Dette vil gøre systemet nemmere at ændre på i fremtiden, eftersom at det kun er controlleren der ændrer på systemets model. Eftersom at Observer pattern også er blevet en del af vores system, har vi valgt at have mere end 1 controller. Dette gør det mere sigende, hvem der skal behandle hvad, frem for at alt datahåndtering ligger i én stor klasse. Dette vil blive forklaret yderligere i afsnittet GRASP.[i]

Singleton pattern (Anders)

Vi har til alle vores controllere valgt at anvende singleton-pattern. Idéen opstod tidligt i udviklingsforløbet, da vi indså vi havde brug for kun at lave én instans af hver controller, og så nemt kunne genbruge denne fra flere klasser. Et eksempel på implementationen finder man i KundeController klassen, som set i Figur 1.

```
13 public class KundeController {
14     private static KundeController inst = null;
15     private CPRnummer cprnummer;
16     private Kunde kunde;
17     private boolean kundeFundet = false;
18     private LinkedList<FFSObserver> observerListe = new LinkedList<>();
19
20     public static KundeController instance() {
21         if (inst == null)
22             inst = new KundeController();
23         return inst;
24     }
25
26     private KundeController() {
27     }
28
```

Figur 1 Eksempel på singleton implementation

Et af kendetegnene er den private contructor, der kun tillader klassen selv at oprette en instans. Constructorkaldet sker i instance() metoden, der, hvis ikke der allerede findes en instans, laver constructorkaldet, og returnerer dette. Hvis objektet allerede er blevet instantieret, returneres instansen.

Et eksempel på instantieringen finder vi i LånetilbudPanel, som vist i Figur 2.

```
43     private KundeController kController = KundeController.instance();
```

Figur 2 Instantiering af singleton

I KundePanel linje 43 finder vi præcis den samme linje, der også her forsøger at instantiere. Uanset hvad der får held med at oprette instansen først, har vi nu to klasser, som begge har tilgang til den samme controller, og dermed den samme datakerne. Dette anvendes blandt andet i LånetilbudController, hvor vi får adgang til den allerede oprettede KundeController, og dermed let kan få returneret de instansvariabler der ligger gemt. I nedenstående tilfælde er det en instans af Kunde vi ønsker (se Figur 3, kodelinje 68).[ii]

```
62    public void beregnLånetilbud(int kunde_id) {
63        this.kunde_id = kunde_id;
64        kreditFundet = false;
65        renteFundet = false;
66        Kreditværdighed kv = new KreditværdighedImpl();
67        KundeController kc = KundeController.instance();
68        Kunde kunde = kc.getKunde();
69        CPRLogik cl = new CPRLogikImpl();
70        CPRnummer cp = null;
71        try {
72            cp = cl.listCPR(kunde.getCPR_id());
73        } catch (SQLException e) {
```

Figur 3 Eksempel på anvendelse af singleton

Observer pattern (Lasse)

Observer pattern, også kaldet publisher-subscriber pattern, er et design mønster som overordnet går ud på at få viderebragt information om, at der er sket ændringer. Måden det foregår på er, at man har en observer, som kan observere på noget (subject), og derved få besked om ændringer. Rent praktisk sker dette i 3 faser:

1. Observeren tilmelder sig hos subject
2. Subject meddeler observeren om tilstandsændringer hver gang det sker
3. Observeren framelder sig som observer på subject

I observer pattern findes der to forskellige teknikker til at oplyse om tilstandsændringer, push og pull. Vi har valgt at bruge push, som går ud på at subjectet både sender en reference af sig selv som parameter, og oplysninger om hvad der er sket. Pull går ud på at observeren selv aktivt skal spørge om det er sket noget.

Vi har i vores projekt anvendt idéen fra observer pattern om, at kunne få besked om når der er sket ændringer. Vi har lavet vores egen FFSObserver, som er et interface, med 1 enkelt metode som hedder update. Update tager 2 parametre, den første som er et Object, det andet som er en String. Object bruger vi til at kunne finde ud af hvilken af vores konkrete observere der har givet besked. String parameteren bruger vi til at finde ud af hvilken metode der er blevet kaldt. På den måde ved vi hvilket subject ændringen er kommet fra, og kan derefter agere ud fra dette. Dette tillader os at foretage små opdateringer i programmet, selvom et enkelt subject kan have mange metoder. Det hjælper også til ikke at overskrive tekstfelter med det samme tekst.

For at kunne få det til at fungere, har vi så lavet 2 metoder i hver af vores controllere, som hedder tilmeldObserver, med en observer som parameter, og en notifyObserver med en String som parameter. tilmeldObserver tilføjer observeren til en liste, hvis den ikke allerede findes i den. notifyObserver kører så listen igennem og kalder update på observerne, med den String som kommer fra notifyObservers.

Det sidste punkt med at framelde som observer har vi set bort fra, da vi kun arbejder med et frame og de samme panels, og mener derfor ikke der skulle være noget behov for at kunne framelde sig.[iii]

Design patterns og arkitektur (Lasse & Thomas)

Det første vi havde i tankerne omkring opbygningen af vores kode var 3-lagsmodellen, som vi har beskæftiget os en del med i vores tidligere projekt. Vi kom så ret hurtigt til en fælles beslutning om, at den også i dette projekt ville være hensigtsmæssig at anvende, specielt med henblik på, at et af kravene til systemet var, at arkitekturen skulle gøre det nemt at flytte til en web platform.

Som forklaret tidligere har vi valgt 3 design mønstre. Vi har selvfølgelig overvejet andre i starten af projektet, og på sin vis også undervejs i forløbet. Facade- og mediatorpattern har været de 2 design mønstre vi har debatteret mest. Disse 2 design mønstre kan være rimelig ens. Idéen vi havde, hvis vi skulle have brugt mediator- og/eller facadepattern var, at vi vidste vi ville have en lang række objekter som skulle kommunikere med hinanden. Dette kunne man løse ved at give dem en mediator at referere til. Nogle ville her gå ind og kalde mediator for en slags facade. Dette ville selvfølgelig løsne koblingen mellem vores objekter, men med 3-lagsmodellen i tanke, syntes vi, at MVC pattern passede bedre.

Unified process (Lasse)

Unified process (UP) er en iterativ udviklingsmetode, som er inddelt i fire faser; inception, elaboration, construction og transition. Faserne har hver deres fokuspunkter, som vil blive beskrevet senere. Faserne køres sekventielt, dvs. man starter altid i inception, og ender i transition. Der er nogle krav om hvad der skal være opfyldt, før man kan gå videre til den næste fase, som vil være fasens milepæl. Hver fase kan inddeles i en til flere iterationer, alt afhængigt af projektets kompleksitet og størrelse. Hver iteration bliver først planlagt, nedbrudt og udført, før man starter næste iteration, vil man så lave iterationsplan for næste iteration.[iv]

Inception

Inception er den fase hvor man starter, og indebærer en opstart af projektet. Projektplanen bliver lavet og visionsdokumentet bliver påbegyndt.

I inception fasen startede vi vores første iteration, iteration 0, hvor vi fik lavet en projektplan, og fik opsat et projekt i Git. Vi valgte fra starten, at bruge vores projektplan som et værktøj til at have et overblik over hvilke opgaver der skulle laves, og inden for hvilken tidsramme. Vi ville ikke ligge vores fokus på hvor lang tid vi brugte på hver enkelt opgave og hvilke dage de blev lavet. Datoerne for hver enkelt opgave er bare autogenereret. Milepælene er faktiske datoer hvor tingene var lavet. Milepælen for fasen er placeret lige før den næste fase.

I iteration 1 valgte vi at placere vores BPR analyse, da det var oplagt at bruge den til at danne os et godt overblik over den nuværende proces, og hvad problemerne ved den var.
Vi fik påbegyndt visionsdokumentet, lavet et use case diagram, casual use cases samt en enkelt fully dressed use case. Vi vurderede at vores fokus skulle ligge på vores UC7, Udarbejd tilbud eftersom at det var den primære funktion i vores system. UC1 og UC2 var også 2 use cases som vi gerne ville lægge fokus på.
Til sidst fik vi lavet et udkast til en faseplan og en iterationsplan for iteration 2.

Milepæl for inception:

- De fleste use cases er identificerede
- Centrale use cases er formelt beskrevet
- Udkast til vision er færdig
- Domænemodel er påbegyndt
- Dataordbog er påbegyndt

Elaboration

I elaboration går fokus mere på analyse og design i forhold til requirements, som ligger hovedsageligt i inception. Det er også her implementationen starter.

Vores elaboration startede med vores iteration 2, hvor vi gik i gang med at udarbejde flere fully dressed use cases og fik lavet casual use cases til. Vi fik startet noget design med klassediagrammer og sekvensdiagrammer. Her gik vores vurdering på, at vi godt kunne nå at implementere UC1 og 2, før vi startede på UC7. Vi startede ud med UC2 opret kunde, og fik lavet diverse diagrammer til den og fik den implementeret. I forbindelse med implementationen har vi testet dele af programmet (se afsnittet om test).

Grunden til, at vi ifølge UP ikke var gået over i construction fasen var, at vi ikke havde fået lavet en supplerende kravspecifikation, som er en milepæl for, at elaboration kunne være gennemført. På trods af denne mangel havde vi været i construction fasen.

Milepæl:

- Alle use cases er identificeret
- De fleste use cases er formelt beskrevet
- Vision er stabil/færdig
- Arkitekturen er velbeskrevet
    - Beskrivelse af ikke-funktionelle krav
    - Eksekverbar prototype
- Projektplan med overblik over iterationer

Construction

Indebærer implementation af resten af systemet med en tilhørende brugervejledning

Milepæl:

- Implementation er færdig
- Systemet består systemtest
- Systemet er stabilt til release
- Brugervejledning er udarbejdet

Transition

Indebærer overdragelsen af softwaren

Milepæl:

- Systemet er leveret og i drift
- Systemet består accepttests
- Brugerne er tilfredse

Discipliner i UP

I forbindelse med vores projekt har vi opdelt alt hvad vi har lavet i mapper med de forskellige discipliner i UP, som bliver beskrevet nedenunder.

Business modelling:

Her har vi placeret vores BPR og vores domænemodel.

Requirements:

Her finder vi artefakter, som omhandler krav til programmet, så som visionsdokument, use cases, operations kontrakter, systemsekvensdiagrammer, aktivitetsdiagrammer og dataordbog.

Design:

I denne disciplin, finder vi klassediagrammer og sekvensdiagrammer samt vores datamodel.

Implementation:

Indebærer alt den kode vi har implementeret i eclipse.

Tests:

Her ligger vores testsuite.

Deployment:

Her ligger vores program og en brugervejledning.

Project management:

Her findes vores projektplan

Environment:

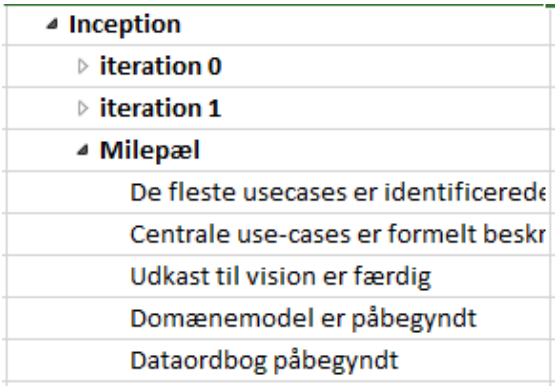
Her ligger filer som opgavebeskrivelse, opslagsværker og diverse hjælpeværktøjer

Projektstyring (Thomas)

I hele projektets forløb har vi forsøgt at holde os til en projektplan.

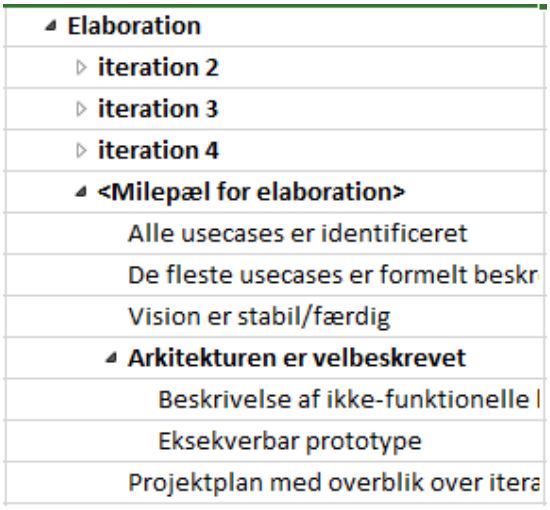
Figur 4

Figur 4 viser vores samlede projektplan, som følger UP. I hver fase har vi lagt et antal iterationer man skal igennem, samt en milepæl for hver, som dikterer hvornår man må gå videre til næste fase.



Figur 5 Eksempel på milepæl

Figur 5 viser den milepæl der skulle være opfyldt før vores inception fase kunne blive afsluttet. Dette skete rimelig tidligt i projektet. Dette skyldes at iterationerne i inception ikke var særligt store.



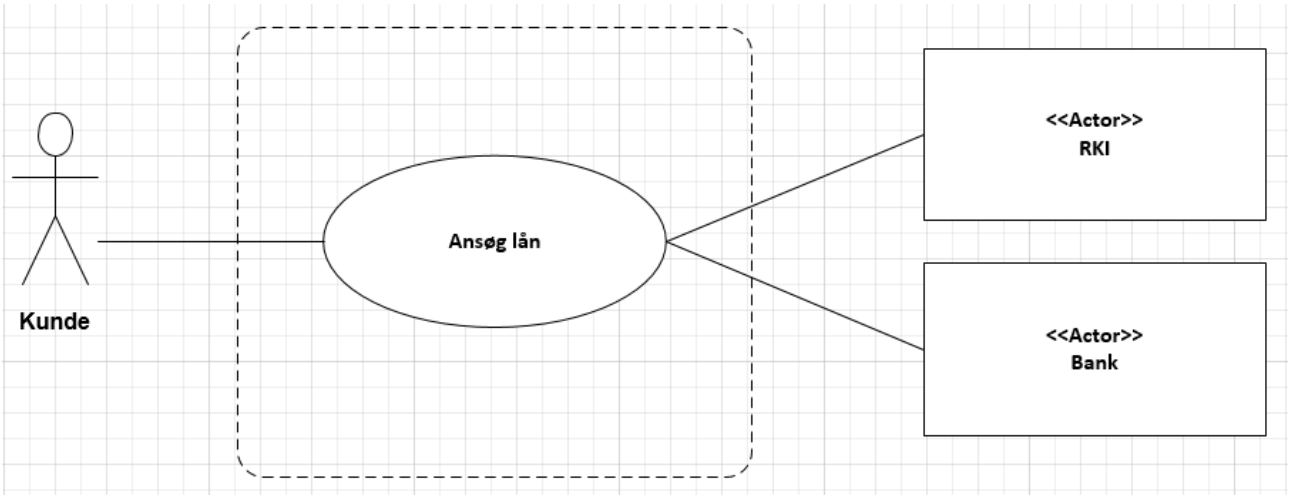
Figur 6 Sidste milepæl

Figur 6 viser den fase vi sluttede i. Elaboration fasen har en meget omfattende milepæl. Man kan selvfølgelig argumentere for, at vi var gået ind i Construction, da vi havde opfyldt et af kravene til fasens milepæl; Systemet er stabilt til release. Vi følte dog at det var vigtigt at vi kunne dokumentere en use case fuldt ud, inden at vi implementerede den, og derfor blev vi i Elaboration fasen indtil slut.

BPR (Simon & Thomas)

Use case diagram

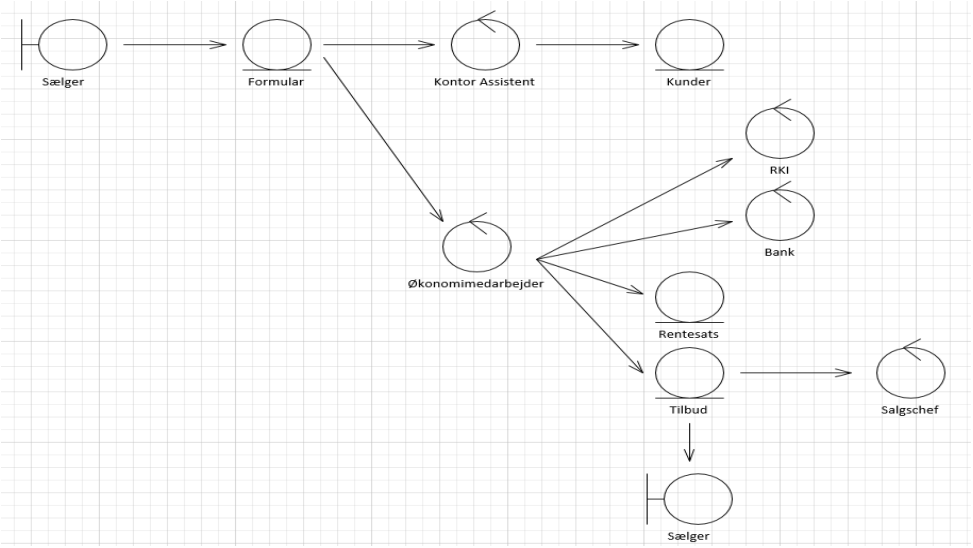
Vi har her lavet et use case diagram (UCD) for virksomhedens perspektiv. Dette giver os en idé om, at RKI og Bank spiller en stor rolle, når en kunde skal ansøge om et lån. UCD for både reverse- og forward engineering ser således ud, da dette ikke burde ændres ved implementationen af vores nye system.



Figur 7 Use case diagram for virksomheden

Reverse Engineering

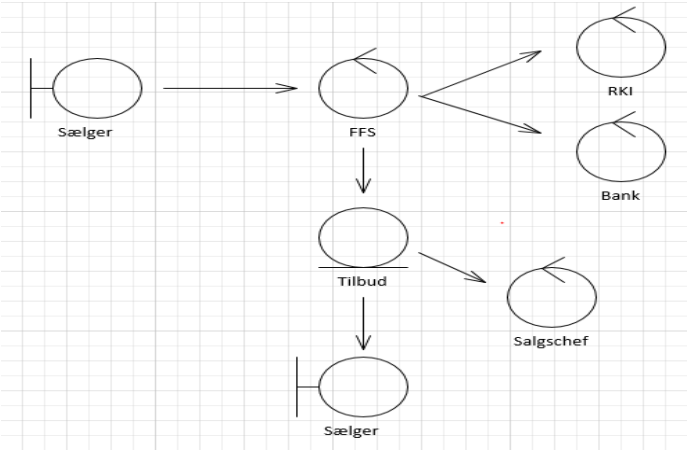
Vi har ved hjælp af reverse engineering, kunne identificere problem domænet, hvilket har givet os en forståelse for, hvordan det fremtidige system skal se ud. Objektmodellen viser at økonomimedarbejderen har ekstremt meget ansvar i det nuværende system, da økonomimedarbejderen skal igennem en masse processer før han/hun kan give lånetilbuddet videre til sælgeren, eller salgschefen, afhængig af lånetilbuddets størrelse, og om kunden har voldt problemer før.



Figur 8 Object model Reverse engineering

Forward Engineering

Vi har ved hjælp af Forward Engineering, kunne løse store problemer ved det nuværende system. Objektmodellen nedenfor viser en tydelig forbedring i forhold til det gamle system. Dette udtrykkes ved at systemet(FFS) er kommet ind i billedet. Ved hjælp af FFS er det muligt at fjerne 2 control-objekter (økonomimedarbejder og kontor assistent). Entity-objektet formular ligger nu i FFS.
FFS står nu for at hente informationer fra RKI og bank. FFS sætter hermed også rentesatsen, så der ikke længere er brug for en medarbejder til at gøre dette. Hermed systematiseres processen betydeligt, hvilket realiserer et hurtigere og mere effektivt system.[v]



Figur 9 Object model Forward engineering

Visionsdokument (Anders, Lasse, Simon & Thomas)

Vision

Vi forestiller os et system, der er baseret på at realisere en kundes drømme uden ventetid. Ferrari Financing System (FFS) har en automatiseret forbindelse til RKI og bank, og kan udregne det bedst mulige lånetilbud for hver enkelt kunde inden for få minutter. Med FFS kan kunden altså få sin drømmebil med hjem den selv samme dag. Derudover vil FFS gøre det muligt for sælgeren at have flere salg med den automatiserede proces. Med vores system vil kunden trygt kunne udlevere sine personlige informationer, da systemet håndterer personlige informationer med diskretion.

Interessentanalyse

Kunden er interesseret i at hans/hendes oplysninger bliver gemt fortroligt, samt at processen ikke tager for lang tid. Endvidere er kunden interesseret i at han/hun kan få det bedst mulige tilbud.

Sælgeren er interesseret i et system der giver hurtig og korrekt respons, for at kunne lave flere salg.

Salgschefen er interesseret i at kunne modtage de lånetilbud der skal godkendes elektronisk. Endvidere er han interesseret i at forøge salget, og sikre høj kvalitet ved udregning af tilbud.

RKI har interesse i at personlige data bliver behandlet med diskretion, og sikkerheden til deres system er vedligeholdt.

Banken har interesse i at sikkerheden til deres system er vedligeholdt.

Ejeren har interesse i at der kan laves lånetilbud på kort tid, med høj kvalitet. Dette kan øge salget.

Datatilsynet har interesse i at personfølsomme oplysninger behandles korrekt og sikkert, i forhold til de gældende regler og love.

Featureliste

Lånetilbud

- Oprettelse
- Kreditværdighed
- Rentesats
- Udregning af rentesats
- Godkendelse
- Eksportering
    - CSV-fil

Kundehåndtering
Persistering af data

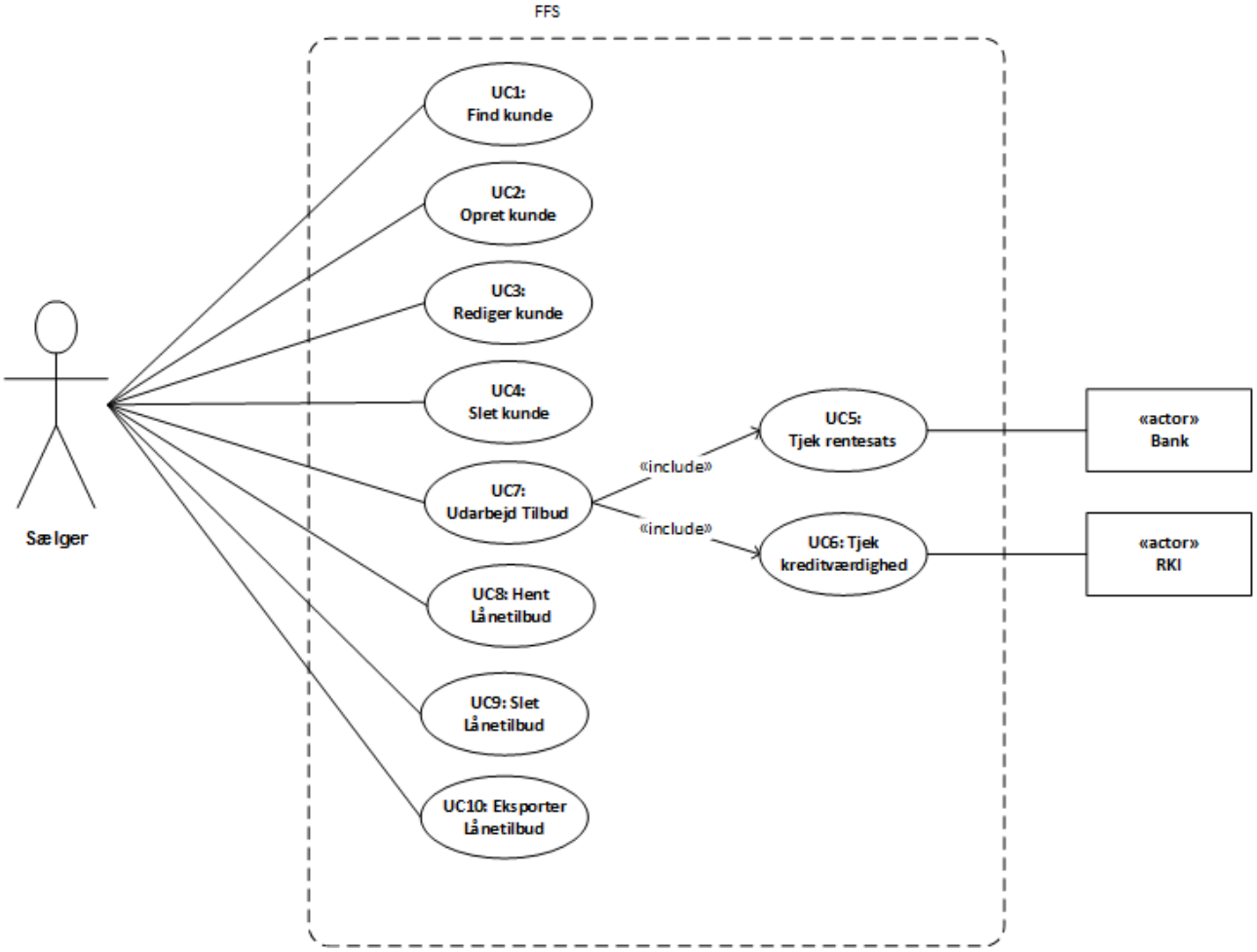
Om visionsdokumentet (Lasse)

Selve visionsteksten er beskrivelsen, som skal sælge systemet til kunden, og overbevise kunden om, at systemet er uundværligt. Derfor er det vigtigt at fokusere på, at visionen ikke er en kedelig opremsning. Ud over visionsteksten indeholder dokumentet også en interessentanalyse, som nævner hvem der kan have interesse i systemet, og hvad de har interesse i. Til sidst er der en featureliste, som viser funktionerne i systemet i en rækkefølge med de vigtigste features øverst. [vi]

Use case diagram (Lasse & Simon)

Et use case diagram (UCD) er en artefakt, som man anvender til at få et overblik over alle use cases (UC), som er i systemet. Ligeledes giver det et overblik over hvem der har med de forskellige use cases at gøre, altså aktørerne. Der findes både primære- og sekundære aktører. Den primære aktør vises altid til venstre i et UCD, og er den som anvender den enkelte use case. Den sekundære aktør vises altid til højre i et UCD, og kan for eksempel være et udefrakommende system som ens eget system interagerer med, for at kunne udføre en given use case.

I et use case diagram arbejdes der med 2 forskellige slags use cases; konkrete- og abstrakte use cases, hvor den konkrete er den, som bliver startet af en aktør, og den abstrakte som startes af en anden use case. [vii]



Figur 10 Use case diagram for systemet

I vores UCD over FFS har vi en primær aktør, sælger, fordi det er ham der anvender vores system og ham der interagerer med hver enkelt use case. Dette kan ses ved at der er en streg fra sælger til hver enkelt use case. Alle de use cases, som er i forbindelse med sælgeren er derfor konkrete use cases og de to sidste UC5 og UC6 er abstrakte use cases. Udover at være en abstrakt use case, har UC5 også en sekundær aktør RKI hvor UC6 har Bank som sekundær aktør.

Fully dressed use case (Simon)

I løbet af vores projekt har vi lavet use cases, de er med til at give os et overblik over funktionelle og ikke funktionelle krav til systemet. Vi har haft stor udbytte af use cases, især da vi fik beskrevet main success scenario og extentions, da det blev lettere at kode efterfølgende.

Main Succes Scenario

1. Sælgeren angiver det samlede lånebeløb.
2. Sælger angiver den ønskede udbetaling.
3. Sælger angiver længden af tilbagebetalingsperioden.
4. Sælger beder systemet om at udregne tilbud.
5. Systemet tjekker den aktuelle rentesats.
6. Systemet tjekker kreditværdighed.
7. Systemet udarbejder tilbud
8. Systemet gemmer lånetilbud.

Extensions

4a Hvis udbetaling er under 50%

1. Tilbud tillægges +1 procentpoint
2. Fortsæt fra hovedscenarie pkt. 5

4b Hvis tilbagebetalingsperioden overskrider 3 år

1. Tilbud tillægges +1 procentpoint.
2. Forsæt fra hovedscenarie pkt. 5

Figur 11 Uddrag af fully dressed use case

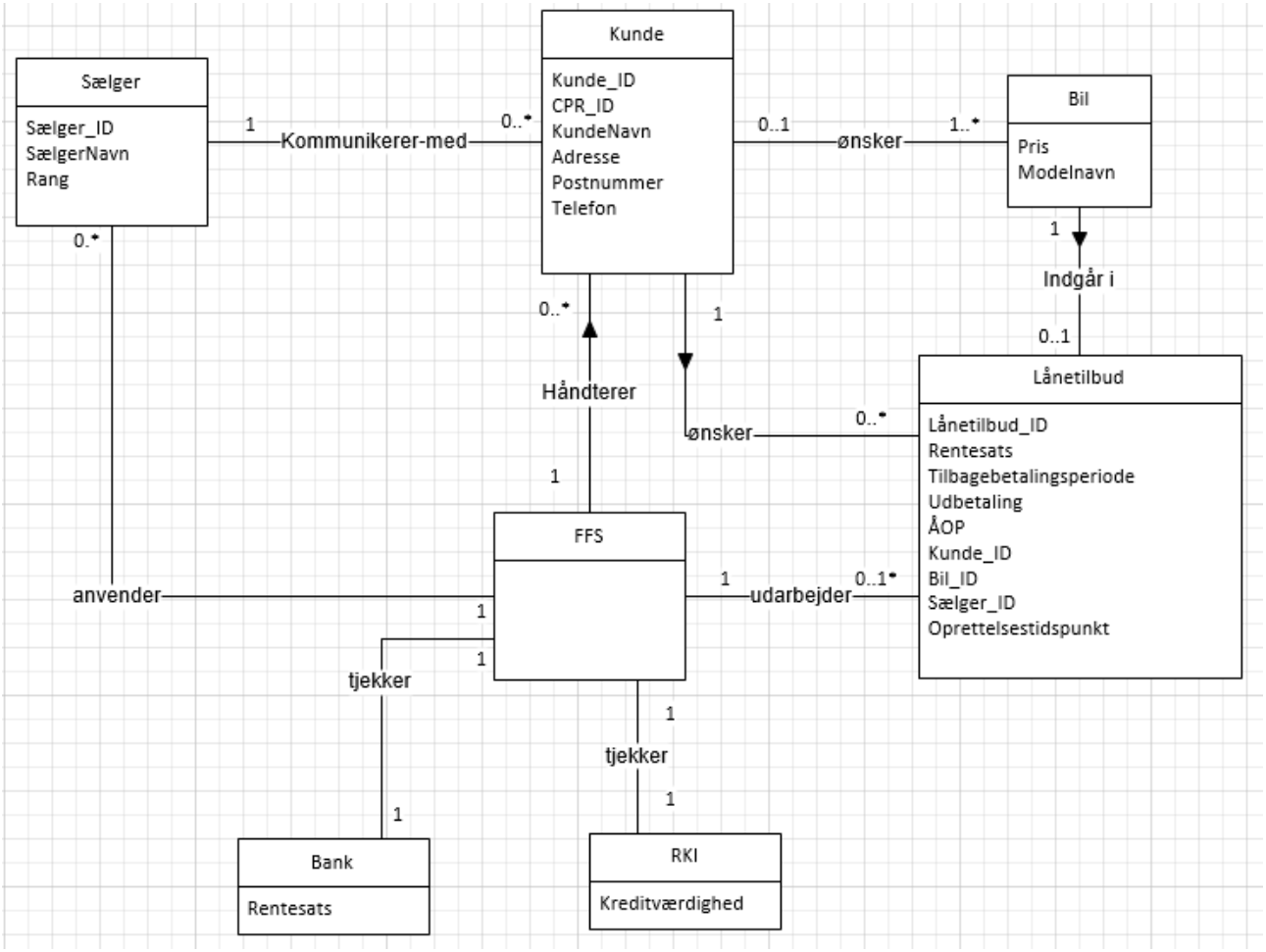
Figur 11 viser et udklip fra vores use case 7 Udarbejd Tilbud. Her har vi identificeret nogle extensions til use casen som påvirker vores system. Det gjorde at vi tidligt i forløbet fik et godt overblik over hvad der skulle implementeres i systemet og hvordan vi kunne gøre det.

```
132     if(udbetaling < pris/2)
133       renteSats += 1;
134     if(tilbageBetalingsPeriode > 36)
135       renteSats += 1;
```

Figur 12 Implementation af use case

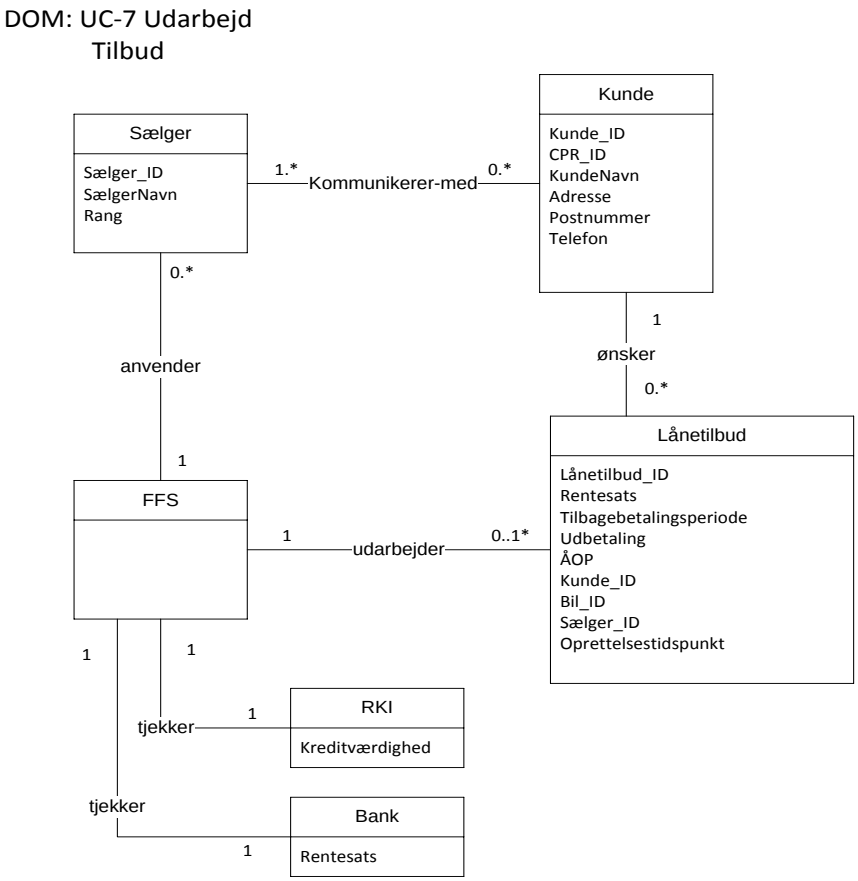
Figur 12 viser to simple if sætninger, som tjekker på henholdsvis udbetaling og tilbagebetalingsperioden. Vi tjekker om udbetaling er mindre end prisen divideret med 2. Hvis den er det lægger vi 1 til rentesatsen, som vi bruger senere til at udregne renten. Det samme sker for tilbagebetalingsperioden, her tjekker vi om den er på mere end 36 måneder, hvis den er det, tillægges der igen 1 til rentesatsen.

Domænemodel (Simon)



Figur 13 Domænemodel for hele systemet

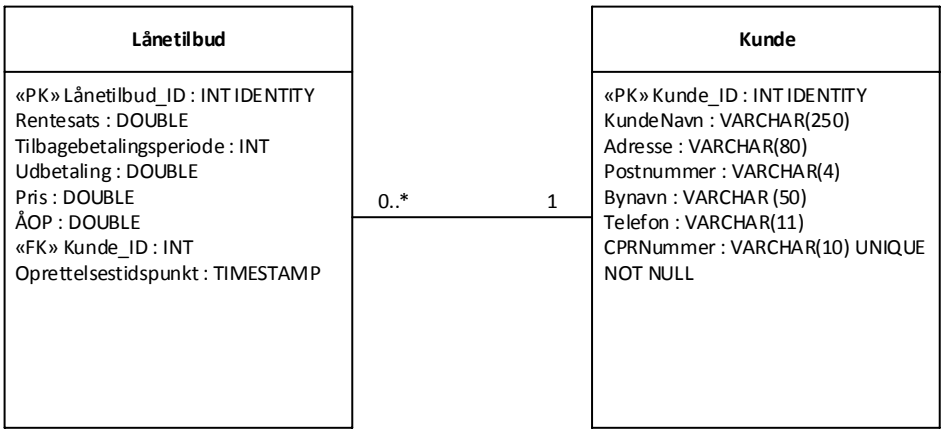
Domænemodellen for det samlede system, har vi lavet for at give et klart overblik over problemdomænet. Vi ser tydeligt at FFS skal håndtere, udarbejde og anvende en masse forskellige ting. Domænemodellen giver os et overblik over hvordan strukturen vil komme til at se ud.



Figur 14 Domænemodel for UC7

Vi har valgt at vise domænemodellen for UC7, da det er en af de centrale use cases for vores system.
I denne use case ser vi hvordan en sælger har kontakt med 1 eller flere kunder, som ønsker et lånetilbud. Sælger anvender derefter systemet (FFS), til at udarbejde et lånetilbud til kunden. Systemet har kontakt til RKI og bank som returnerer henholdsvis Kreditværdighed og Rentesats til systemet.[viii]

Argumentation for 3. normalform (Anders)



Figur 15 Datamodel tidlig udgave

Tidligt i forløbet så vores datamodel således ud. Dette var første udkast, og primært brugt til et overblik over hvilke værdier der skulle være i databasen. Vi udvidede hurtigt modellen til at inkludere information omkring sælger og bil, med dertilhørende værdier. Disse to blev koblet på lånetilbud med en foreign key/primary key relation. Dette tillod os at have tabeller med sælgere og biler oprettet, og blot linke dem på når et nyt lånetilbud blev lavet. Modellen levede stadig ikke op til de 3 normalformer. 1. normalform er bestået, da alle attributter kun har én værdi.

Det første problem lå i bynavn og postnummer, da bynavnet er fuldstændig afhængigt af postnummeret. Disse to blev derfor sat i en tabel for sig selv, og så lod vi postnummeret i kundetabellen referere til postnummeret i postnummer tabellen. Således havde vi altid både postnummer og bynavn tilgængeligt, såfremt vi kendte postnummeret.

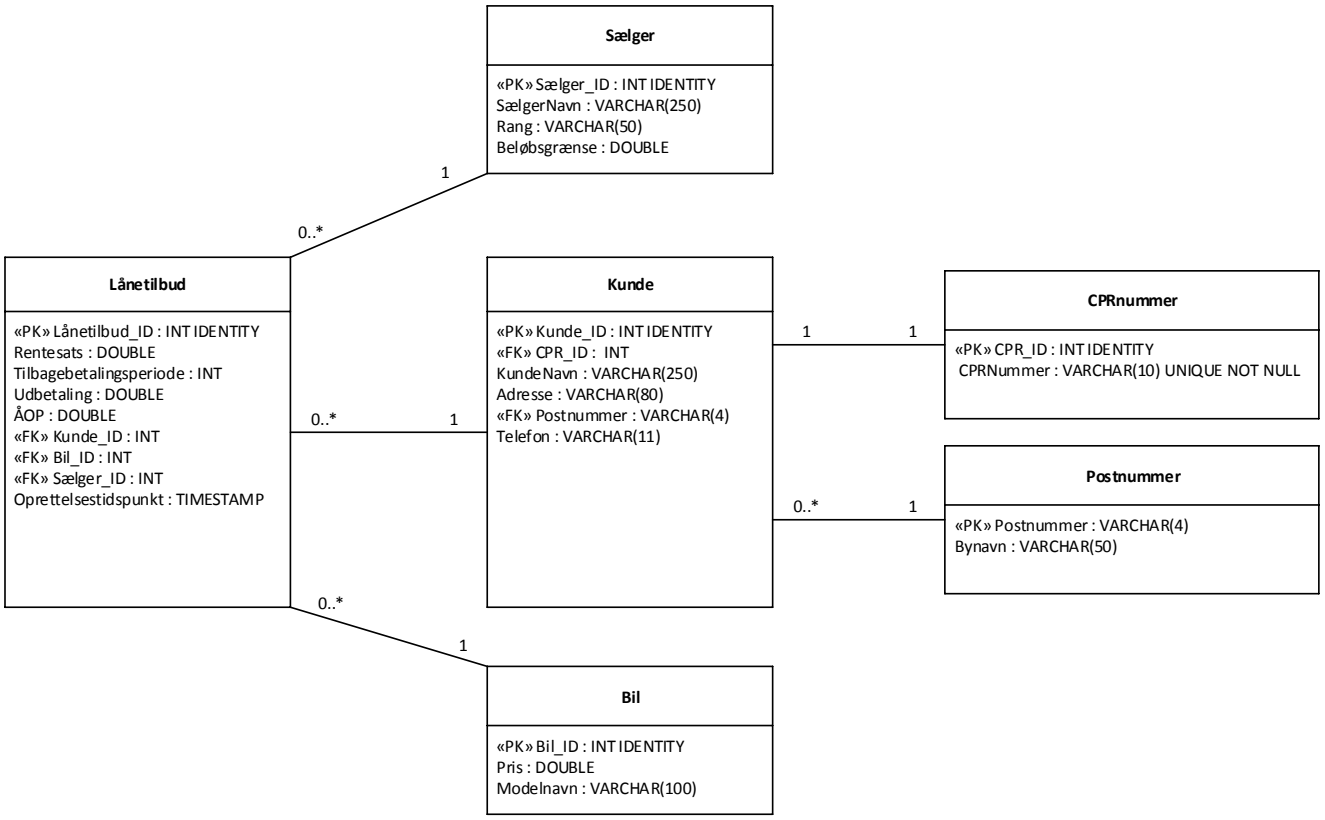
Efter at vi havde flyttet pris ud i en tabel sammen med de nye attributter kunne modellen bestå 2. normalform, da alle attributter nu var fuld afhængige af den primary key der ligger i den tilsvarende tabel.

Til sidst valgte vi at lade CPRnummer ligge i en tabel for sig selv, med en dertilsvarende CPR_id, der refererede tilbage til kunde tabellen. Det kan argumenteres at datamodellen allerede her opfyldte den 3. og sidste normalform, da der ikke direkte var nogle transitive afhængigheder. Det blev dog besluttet at flytte CPRnummer ud, da dette evt. i en fremtidig revision kunne øge sikkerheden. Den opdaterede datamodel kan ses på Figur 16 Datamodelnedenfor.

Endelige overvejelser (Anders & Lasse)

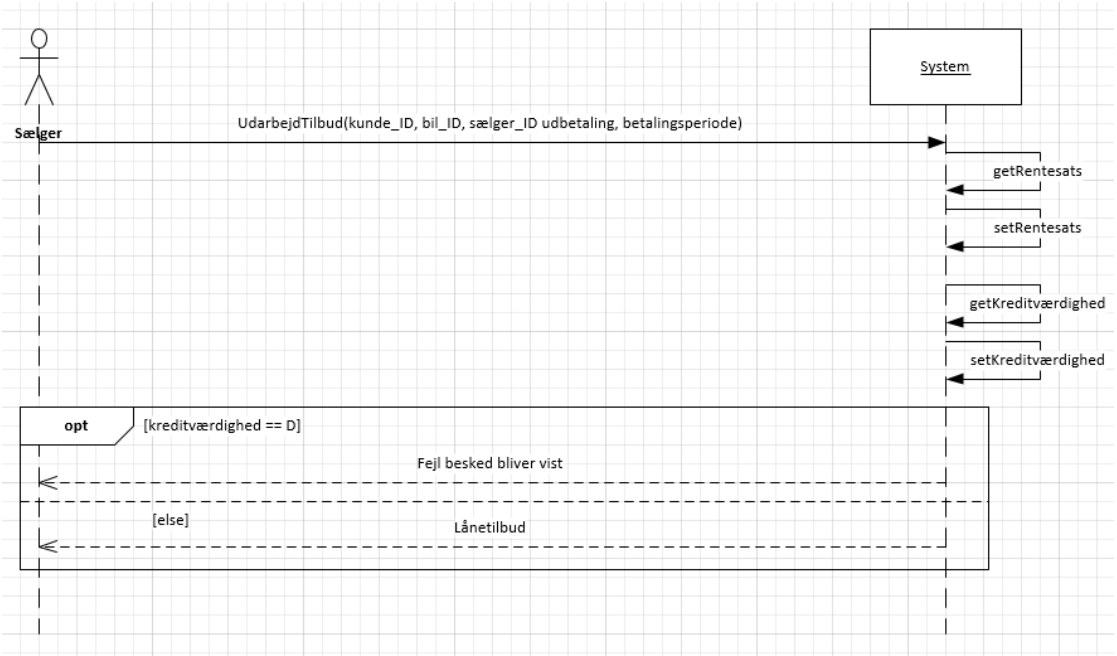
Databasen har et begrænset omfang, og der er helt klart plads til forbedringer. Vi har foretaget nogle valg, der måske går imod visse standarder, for at holde det simpelt. Bil tabellen indeholder kun attributterne pris og modelnavn. I et virkeligt system ville dette ikke være tilstrækkeligt, da der ikke er plads til fx tilvalg eller tilbud.

Ligeledes findes der i sælgertabellen en attribut; beløbsgrænse. Denne *burde* være i en tabel for sig selv, forbundet til sælgertabellen med primary-/foreignkey relation via rang. Vi har valgt at begrænse vores system til ganske få sælgere, og derfor tillod de enkelte gentagne værdier i samme kolonne der nu måtte være. Det betyder selvfølgelig at det ikke overholder normalformerne til fulde. Eftersom at vi arbejder med hardcodede værdier, som vi ikke skal ændre i, vil det ikke komme til at have nogen indflydelse.



Figur 16 Datamodel

Systemsekvensdiagram (Thomas)

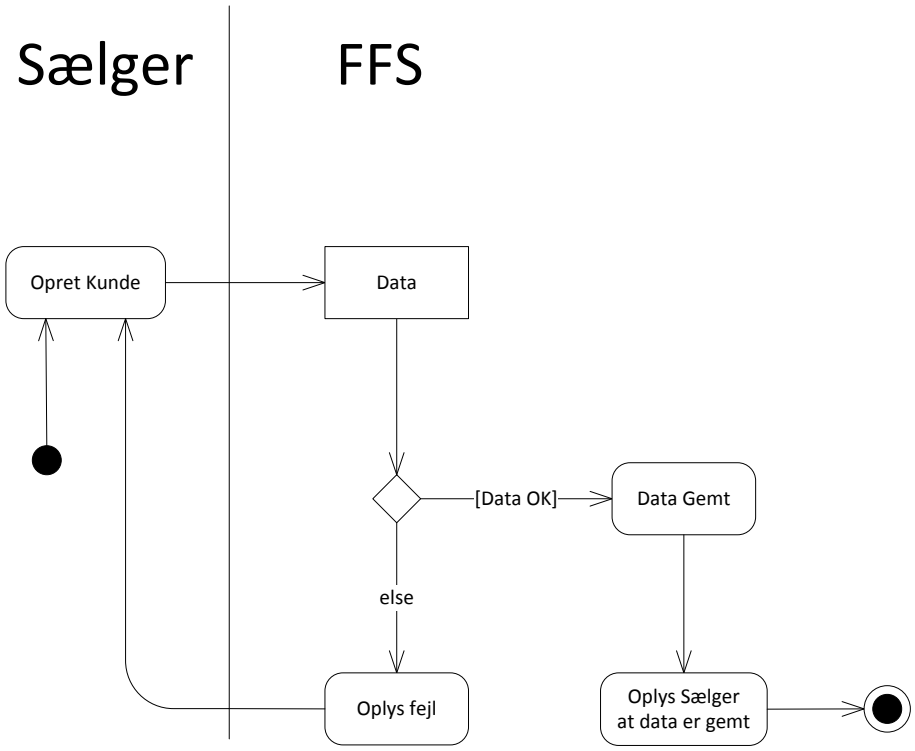


Figur 17 Systemsekvensdiagram for UC7

Systemsekvensdiagrammet for UC7 – Udarbejd tilbud, går i sin store helhed ud på, at systemet skal have en kunde, bil, sælger, et beløb, samt en betalingsperiode, for at kunne fortsætte processen. Vi var ved dette tidelige stadie klar over, at en rentesats og en kreditværdighed, skulle findes før et tilbud kunne beregnes. Metoden kaldet Udarbejd tilbud sender de nødvendige parameter med, og herefter starter systemet med at finde rentesatsen på den givne dag, og finde kreditværdigheden på kunden, som blev sendt med som parameter. Systemet tjekker om kunden har kreditværdighed D. Hvilket resulterer i en fejl besked, da systemet allerede her ved, at kunden ikke vil få lånet godkendt. Hvis kunden har kreditværdigheden A, B eller C, fortsætter systemet med beregningerne, og sender så tilsidst et lånetilbud tilbage til sælgeren.[ix]

Aktivitetsdiagram (Simon)

I et aktivitetsdiagram tager vi en use case og visualiserer for os selv hvordan processen forløber. I et aktivitetsdiagram har man den primære aktør til venstre og systemet til højre som så er opdelt med en linje ned i midten.



Figur 18 Aktivitetsdiagram UC

Man starter fra den sorte prik(Found) og så bevæger man sig ellers bare med pilene. Den roterede firkant som deler pilen i to, kaldes "decision". Her kigger man på to muligheder, altså en if/else. Efter man har fundet ud af hvilken vej der er den rigtige, fortsætter man så igennem indtil man når den sorte prik med en cirkel omkring(Lost), hvilket er slutningen på aktiviteten.
Diagrammet giver os en god idé om flowet i processen.[x]

Operationskontrakter (Thomas)

UC5 – OC2: setRenteSats	UC6 – OC1: setKreditvaerdighed
Systemoperation	**Systemoperation**
setRenteSats(callback : Callback)	setKreditvaerdighed(cpr : String, callback : Callback)

Figur 19 Dele af operationskontrakt

Ved operationskontrakterne har vi valgt at fokusere på centrale dele af systemet. I use case diagrammet blev der vist, en tydelig forbindelse mellem UC5 og UC6 til UC7. I figur 19 ser vi vores første operationskontrakt, setKreditværdighed. Hvis vi ser på parametrene på denne metode (og for OC2), er der én parameter springer i øjnene; Callback.
Ideen med Callback parameteren var, at vi allerede her vidste at disse to metoder(setKreditværdighed, setRenteSats) skulle køres i tråde. Her set i koden til setKreditværdighed:

```
@Override
public void setKreditværdighed(String cpr, CallBack callBack) {
    Thread thread = new Thread() {
        public void run() {
            kreditværdighed = cR.rate(cpr);
            switch (kreditværdighed) {
            case A:
                tillægspoint = 1;
                kvAcceptabel = true;
                break;

            case B:
                tillægspoint = 2;
                kvAcceptabel = true;
                break;

            case C:
                tillægspoint = 3;
                kvAcceptabel = true;
                break;

            default:
                kvAcceptabel = false;
                break;
            }

            callBack.onRequestComplete();
        };
    };
    thread.start();
```

Figur 20 Eksempel på brug af threads

Callback parameteren tillader os at kalde en metode som hedder onRequestComplete(). Efter at tråden er blevet startet, bliver der oprettet forbindelse til RKI, hvilket så resulterer i en kreditværdighed. Dette bliver så tjekket igennem switch sætningen, hvorefter callback kalder onRequestComplete() på sig selv. Dette gør at systemet ved præcis hvornår metoden setKreditværdighed() er færdig. Ellers skulle systemet sidde og pinge i et fast interval, for at finde ud af hvornår den er færdig(se afsnittet om observerpattern).

```
@Override
public void setRenteSats(CallBack callBack){
    Thread thread = new Thread(){
        public void run(){
            rate = ir.todaysRate();
            callBack.onRequestComplete();

        };
    };
    thread.start();
}

@Override
public double getRenteSats(){
    return rate;
}
```

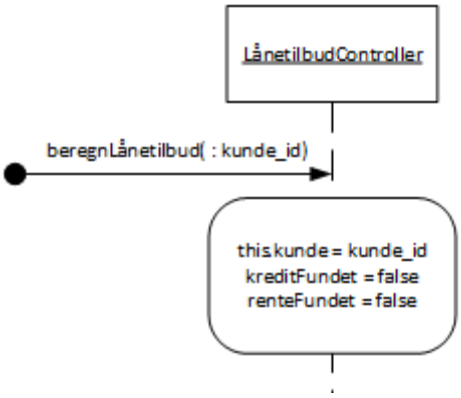
Figur 21 Eksempel på brug af threads

Ovenfor ser vi metoden setRenteSats(). Idéen er den samme som for setKrediværdighed(). Dette gør vores 2 tråde meget mere effektive, eftersom systemet kan fokusere på andre opgaver, indtil at onRequestComplete() bliver kaldt.[xi]

Operationskontrakterne kan ses i deres helhed i bilag 7-11.

Sekvensdiagram (Anders & Thomas)

Sekvensdiagrammet for UC7 viser forløbet fra kaldet i LånetilbudPanel, indtil de to use cases (UC5 og UC6). Metoden den viser ligger i LånetilbudController, linje 62-96. Diagrammet er i sig selv rimelig selvforklarende, men der er alligevel flere elementer af interesse. Nedenstående eksempel viser konsistensen mellem diagram og kode, hvor metodekaldet resulterer i variabelerklæringen.

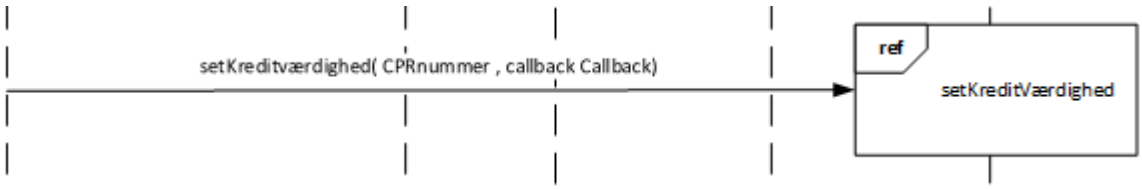



```
62  public void beregnLånetilbud(int kunde_id) {
63      this.kunde_id = kunde_id;
64      kreditFundet = false;
65      renteFundet = false;
```

Figur 22 Kode tilhørende sekvensdiagrammet

Figur 23 Sekvensdiagram UC7 udklip

Videre i diagrammet ligger der en reference til setKreditværdighed, som er videre beskrevet i sekvensdiagrammet for UC6. Noteringen ser således ud:



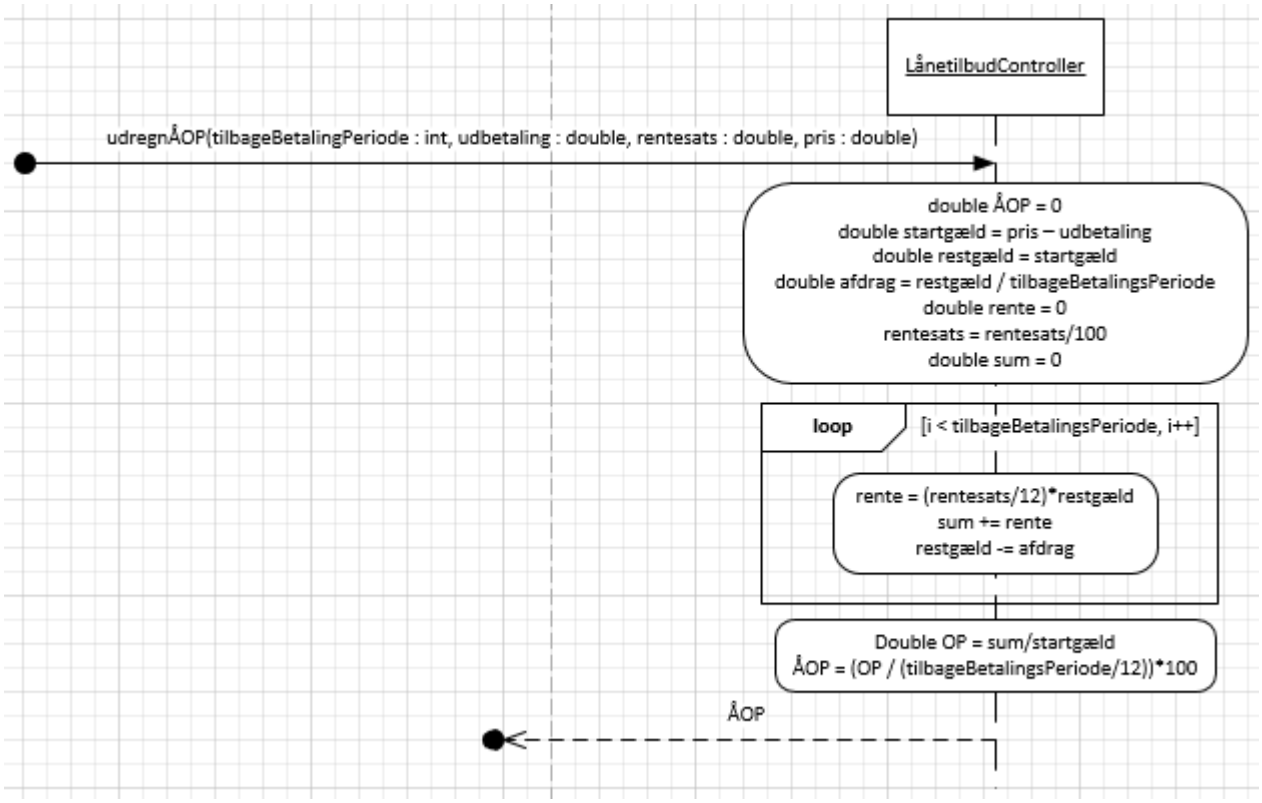
Figur 24 Sekvensdiagram UC7 udklip

I koden kommer det til udtryk med kaldet setKreditværdighed på kv, der er en instans af KreditværdighedImpl.

```
77        kv.setKreditværdighed(cp.getCPRnummer(), new CallBack(){
78            @Override
79            public void onRequestComplete() {
80                kvAcceptabel = kv.getkvAcceptabel();
81                kreditFundet = true;
82                tillægspoint = kv.getTillægspoint();
83                notifyObservers("Kreditværdighed");
84            };
85        });
```

Figur 25 Kode tilhørende sekvensdiagrammet

De to reference sætninger oplyser om de ændringer der er sket, når deres tråde er kørt færdig, og data behandles videre derfra. Sekvensdiagrammet for UC7 slutter derfor med de to referencekald, hvilket forklarer at der ingen returkald eller anden form for afrunding ligger bagefter.



Figur 26 Sekvensdiagram UC7 udklip

Sekvensdiagrammet for udregnÅOP viser primært hvordan vi kommer frem til ÅOP. Der bliver instantieret en række lokalvariabler i starten. Men det interresante sker i loopet. I loopet finder vi summen, som vi bruger til at udregne OP(Omkostnings procent). Dette gør vi ved at køre en for-løkke igennem, for at finde frem til summen. For-løkken kører én gang for hver måned tilbageBetalingsPeriode er blevet sat til. Vi ender så ud med at få OP, som vi skal bruge i formlen til ÅOP. LånetilbudController returnerer så til sidst ÅOP til den der har kaldt metoden.

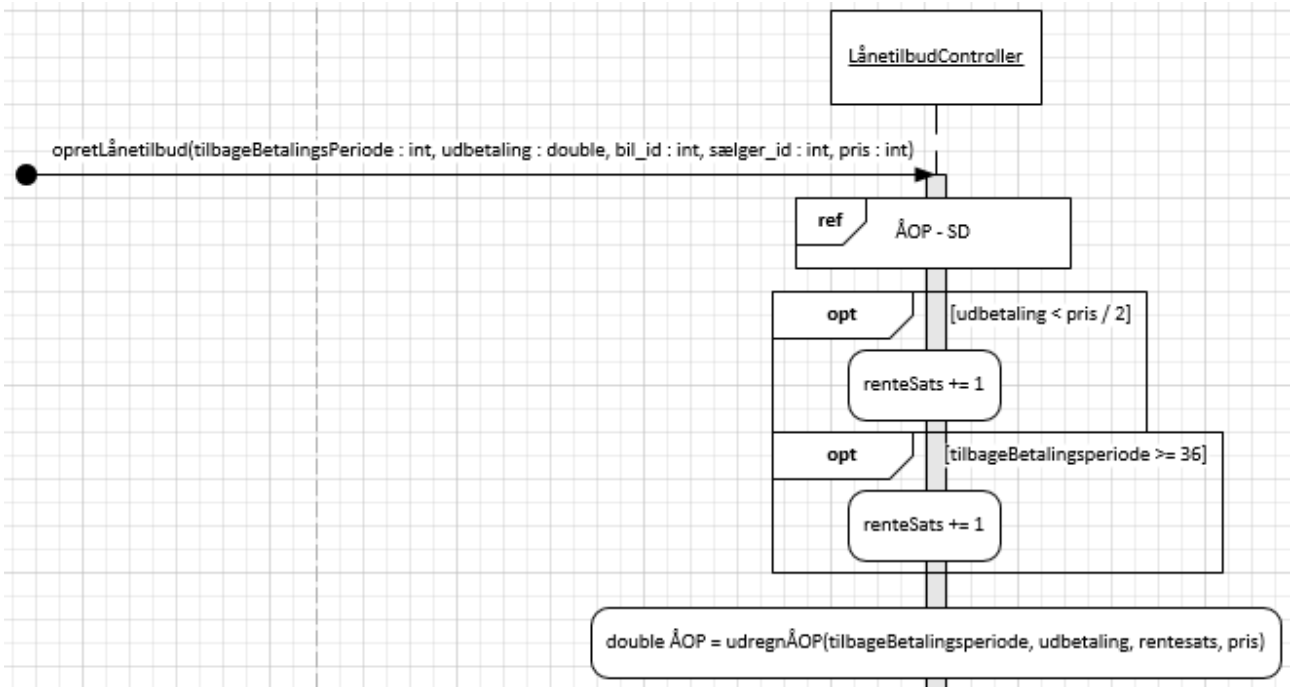
```
public void opretLånetilbud(int tilbageBetalingsPeriode, double udbetaling, int bil_id, int sælger_id, double pris){
    java.util.Date date= new java.util.Date();
    Timestamp timestamp = new Timestamp(date.getTime());

    if(udbetaling < pris/2)
        renteSats += 1;
    if(tilbageBetalingsPeriode >= 36)
        renteSats += 1;

    double ÅOP = udregnÅOP(tilbageBetalingsPeriode, udbetaling, renteSats, pris);
    LånetilbudLogik lt = new LånetilbudLogikImpl();
    Lånetilbud lånetilbud = new LånetilbudImpl();
    lånetilbud.setRentesats(renteSats);
    lånetilbud.setTilbagebetalingsperiode(tilbageBetalingsPeriode);
    lånetilbud.setUdbetaling(udbetaling);
    lånetilbud.setÅOP(ÅOP);
    lånetilbud.setKunde_id(kunde_id);
    lånetilbud.setBil_id(bil_id);
    lånetilbud.setSælger_id(sælger_id);
    lånetilbud.setOprettelsestidspunkt(timestamp);
    try {
        lt.createLånetilbud(lånetilbud);
    } catch (SQLException | LånetilbudAllreadyExists e) {
        e.printStackTrace();
    }
    notifyObservers("opretLånetilbud");
}
```

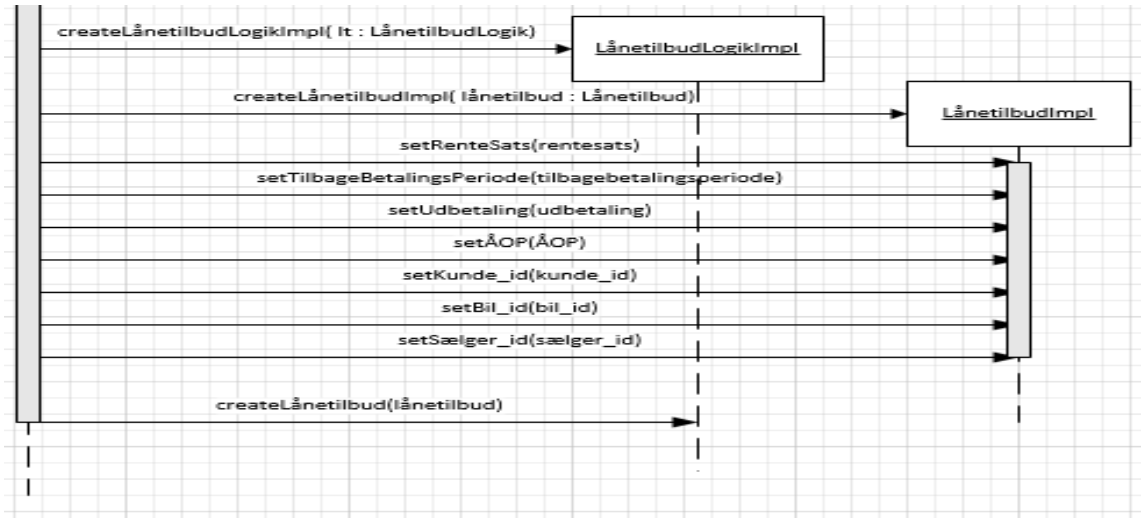
Figur 27 Eksempel fra koden

I Figur 27 kan vi se (markeret med rødt), at udregnÅOP bliver kaldt når ét lånetilbud skal oprettes. Det er også vigtig at bemærke at, rentesatsen der bliver parameter i udregnÅOP, er den daglige rentesats fra banken. Denne rentesats bliver nemlig altid beregnet, før man kalder opretlånetilbud. Dette kan ses i sekvensdiagrammet for beregnLånetilbud.



Figur 28 Sekvensdiagram UC7 udklip

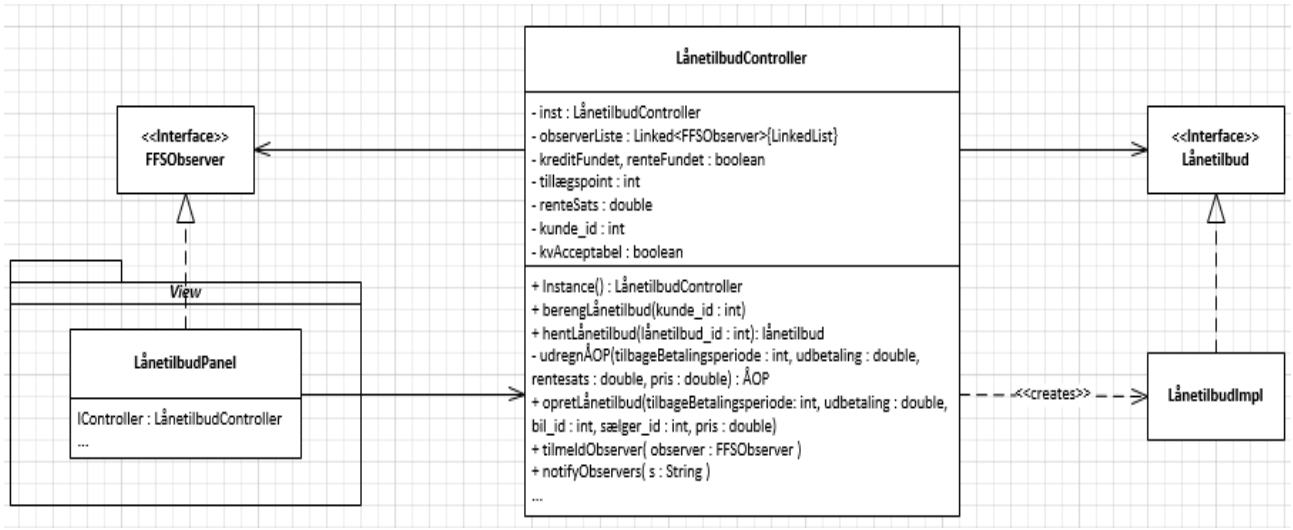
I Figur 28-29, ser vi den ref, vi nævte før. Går vi længere ned af livslinjen, ser vi også at vi tager højde for, hvis udbetalingen er mindre end 50% af den absolute pris, skal der lægges +1 procentpoint til rentesatsen. Dette gælder også hvis tilbageBetalingsperioden er mere en 36 måneder. Når den effektive rentesats er blevet udregnet, bliver udregnÅOP kaldt.



Figur 29 Sekvensdiagram UC7 udklip

Senere på livslinjen bliver der lavet 2 objekter: LånetilbudLogikImpl og LånetilbudImpl. Herefter bliver alle de nødvendige instanser for at lave et lånetilbud sat på LånetilbudImpl. Efter dette bliver der sendt et lånetilbud afsted til LånetilbudLogikImpl, som så lægger lånetilbuddet ind i databasen. Sekvensdiagrammet for opretLånetilbud kan ses i dets helhed i bilag 16.[xii]

Klassediagram (Thomas)



Figur 30 Klassediagram UC7

Vi har valgt at fremhæve klassediagrammet for UC7, hvor vi ser hvordan vores LånetilbudController fungerer. Klassediagrammet giver os et overblik over hvad en LånetilbudController skal kunne, og hvem der skal hjælpe den med at løse problemer. Figur 30 illustrerer en FFSObserver, hvilket giver en klar indikation om, at observer pattern skal anvendes. Ved at associere LånetilbudController med interfacet FFSObserver, gør vi det muligt at observere om der sker noget i de andre observere. Dette gør det muligt for LånetilbudController at beregne et lånetilbud.

```
public void update(Object source, String s) {
```

Figur 31 Update metode i observer

Vi ser her update metoden, som ligger i vores LånetilbudPanel. Denne metode bliver overrided fra FFSObserver interfacet. Dette gør at når LånetilbudController kalder notifyObservers(), bliver metoden update kaldt i LånetilbudPanel. Metoden update gør en masse forskellige ting, ud fra hvem det er der har kaldt den.

```
else if (source instanceof LånetilbudController) {
  if ((s.equals("RenteSats") || s.equals("Kreditværdighed")) && lController.beggeFundet()) {
      if(lController.getkvAcceptabel()){
          int tilbageBetaling = Integer.parseInt(tfTilbagebetalingsperiode.getText());
          double udbetaling = Double.parseDouble(tfUdbetaling.getText());
          int bil_id = bController.getBil().getBil_id();
          int sælger_id = sController.getSælger().getSælger_id();
          double pris = bController.getBil().getPris();
          lController.opretLånetilbud(tilbageBetaling, udbetaling,bil_id, sælger_id, pris);
      }else{
          JOptionPane.showMessageDialog(null, "Kundens kreditværdighed er ikke tilstrækkelig","Fejl!", JOptionPane.ERROR_MESSAGE);
          loadingIcon.setVisible(false);
          btnBeregnLånetilbud.setEnabled(true);
      }
```

Figur 32 Update metode i controller

I Figur 32 fokuseres der på LånetilbudController, og hvad der sker hvis den kalder notifyObservers(). I den første if sætning tjekker vi om notifyObservers er blevet kaldt med ”RenteSats” eller ”Kreditværdighed”, og tjekker bagefter om både RenteSats og Kreditværdigheden er fundet, da dette er en betingelse, for at kunne fortsætte processen. Herefter finder vi ud af om kreditværdigheden er acceptabel, hvis den, er fortsætter vi. Derefter bliver en masse variabler instansieret, herefter kalder update metoden videre på LånetilbudController, hvor opretLånetilbud bliver kaldt, med de værdier den har fået fra de andre controllere.
Vores update metode kan ses i LånetilbudPanel klassen (view pakken) fra kodelinje 340.[xiii]

GRASP (Anders)

Et gennemgående koncept i udviklingen af vores system har været evnen til at kunne videreudvikle og/eller genbruge væsentlige dele af programmet. Vi har blandt andet valgt at anvende interfaces til de fleste klasser, for at gøre det lettere at udskifte/opdatere de dertilhørende klasser. Desuden har vi oprettet flere controllere, der hver især fungerer som information experts. Til funktionalitet på lånetilbud findes der derfor en lånetilbudcontroller, der håndterer logikken (der derved fjerner alt logik fra viewet) for lånetilbud. Vi har fulgt dette system igennem, med den eneste undtagelse af CPR numre. Logikken til dette ligger i kundecontrolleren, da vi mente en CPRcontroller ville blive for lille, og alligevel være koblet til kundecontrolleren. Et kig på importlisten for KundeController viser også hvilke forbindelser denne har til andre klasser

```
6  import domain.CPRnummer;
7  import domain.CPRnummerImpl;
8  import domain.Kunde;
9  import domain.KundeImpl;
10 import exceptions.CPRAllreadyExistsException;
11 import exceptions.KundeAllreadyExistsException;
```

Figur 33 Importliste for KundeController

KundeController anvendes primært af KundePanel, der ligesom controlleren har begrænset sit import til de relevante klasser. Ud over en længere række GUI imports ser listen således ud:

```
25 import logik.FFSObserver;
26 import logik.KundeController;
27 import logik.PostnummerController;
28 import exceptions.CPRAllreadyExistsException;
29 import exceptions.KundeAllreadyExistsException;
30 import exceptions.PostnummerDoesNotExistException;
```

Figur 34 Importliste for KundePanel

Postnummer er ligesom CPR tæt knyttet til kunde, men der ligger alligevel lidt logik for sig selv, derfor har denne fået sin egen controller. Dette gør det ikke bare muligt at udskifte dele af eller hele controlleren, men også nemt at finde frem til den præcise metode man har brug for.

Kobling (Thomas)

I vores system har vi selvfølgelig gået efter og få den mindst mulige kobling imellem objekter, dog stadig haft i tankerne at for lav kobling kan føre til forvirring i det samlede billed af systemet. Da en vis kobling imellem objekter aldrig kan undgås har vi, som nævnt tidligere, benyttet os af interfaces og controllers. Dette har selvfølgelig noget at gøre med hvilke design patterns vi har valgt at gå ud fra. Ved brug af disse interfaces har det lykkedes os, at skabe dependencies, hvilket er den svageste kobling, fremfor nedarvning som er den stærkeste. Dette har dog været nødvendigt i vores tests. Det er også blevet brugt i vores GUI kode, men her har nedarvningen været af JPanels osv.

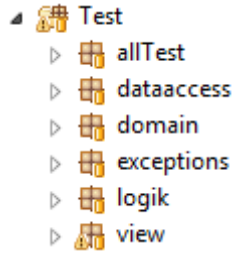
Samhørighed (Anders)

Som tidligere nævnt har en høj samhørighed været i højsædet under udviklingen. Vi har så vidt muligt begrænset hver klasse til det relevante scope. Således kan der trækkes en lige linje fra KundePanel til KundeController, via KundeLogik til Kunde i domain.

For at mindske koblingen ligger der både metoder til at oprette kunde og finde kunde i KundeController. Disse metoder har som sådan ikke noget med hinanden at gøre, og for at øge samhørigheden kunne disse være lagt i klasser for sig selv. Vi har dog valgt at lægge dem sammen, dels for at undgå for små controller klasser, dels for at have en høj samhørighed ved controllere.[xiv]

Test (Thomas)

Vi har i vores projekt testet vores system undervejs i processen. Dette har vi gjort på en systematisk måde, så hvis vi senere i processen skulle ændre noget i koden, skulle testene stadig kunne køre. Vi har valgt at fokusere på centrale dele af systemet, og lave test suites til disse. Grunden til at hele systemet ikke bliver testet, er at man kan ende med at dobbelt teste bestemte dele af systemet, og vi ikke har set det nødvendigt og gøre. Vores test har hjulpet os til en bedre forståelse for dele af vores kode, samt også gjort det muligt for os, at identificere fejl tidligere. De centrale dele vi har fokuseret på er UC5, UC6 og UC7.[xv]



Figur 35 Test folder

Figur 35 viser vores test folder. Vi har delt det op på denne måde, alle test metoder i domain package tester alle metoder i domain package i src folderen. I hver package har vi også en TestSuite for alle test klasserne i den package. De bliver så alle sammen samlet i allTest, som er en TestSuite til alle vores tests.

```
@Test
public void testSetKreditvaerdighed() throws Exception {
    int timeOut = 5000;
    kv.setKreditværdighed("3006921611", new CallBack(){
        @Override
        public void onRequestComplete() {
            requestCompleted = true;
        };
    });
    while(!requestCompleted && timeOut > 0){
        Thread.sleep(500);
        timeOut -= 500;
    }
    if(requestCompleted)
        assertEquals( Rating.valueOf("B"), kv.getKredigværdighed());
        assertEquals(true, kv.getkvAcceptabel());
        assertEquals(2, kv.getTillægspoint());
    if(requestCompleted == false)
        fail("Call to RKI exceeded timeout");
}
```

Figur 36 Eksempel på test klasse

Figur 36 viser koden for vores KreditværdighedTest klasse. Dette er den eneste metode der bliver kørt i denne klasse. Dette skyldes at der kun skal testes på en ting i vores Kreditværdighed klasse. Om den gør det rigtige, ud fra hvilken kreditværdighed kunden nu har. Der bliver instansieret 2 private variabler i denne klasse også. kv som er af typen Kreditværdighed, samt en boolean kaldet requestcompleted.
Vi opdagede at det var nødvendigt for os at lave en variabel i metoden, som vi kalder timeOut. Dette skyldes at hvis vi kørte metoden uden vores timeOut, ville metoden simpelthen fejle, da vores kreditværdighed metode der bliver kørt, er en Thread. Denne Thread eksisterer, da vi både skal kunne hente kreditværdighed og rentesats, uden at skulle vente på at den anden blev færdig. Vi tæller så timeOut ned i metoden så længe requestCompleted er false, og timeOut er størrere end 0. Vores onRequestComplete() er en metode der bliver overrided fra Kreditværdighed klassen.
Vi forventer så i vores assertEquals, at kreditværdigheden er ”B”, at kunden er acceptabel for lån, og at der bliver lagt 2 tillægspoint til vores tillægspoints variable. Hvis dette fejler, vil der blive skrevet en fejl besked ud.
Testen for RenteSats er for sin vis den samme. Metoderne til disse to klasser har hjulpet os med og forstå hvordan vores threads virker, og ikke mindst hvordan vores CallBack variable virker.

```
@Test
public void testValiderCPR() {
    assertEquals(false, oks.validerCPR("asdsbdfsbas"));
    assertEquals(true, oks.validerCPR("3006921811"));
}
```

Figur 37 Eksempel på view test

Vi har selvfølgelig også testet på vores view. Her har vi fokuseret på at teste vores valider metoder, og om de virkelig kan validere input fra brugeren. Klassen tester alt fra CPRnummer, til navn på kunden. Denne klasse kan ses i sin helhed i view package i vores Test folder.
Test suites for systemet vil være og finde i bilag.

Dataordbog (Simon)

I vores dataordbog finder man definitioner og eksempler på domæneterminologi. Væsentlige termer bliver beskrevet i dette afsnit, det samme gælder med forkortelser. Der er to forskellige definitioner i dataordbogen. Den ene er der intensionel definition som giver en meget generel beskrivelse af det givne koncept. Den anden definition er den ekstentionelle definition, som giver et mere konkret eksempel på konceptet. I forhold til objekt orienteret programmering vil klasser være lig med den intensionelle definiton og objekter vil være lig med den ekstentionelle definition.[xvi]

Årlig omkostning i procent (ÅOP)

Intensionel definition

ÅOP er hvad navnet ligger op til, årlig omkostning i procent. Gebyrer, renter og andre udgifter bliver lagt sammen her så man får et nemt og overskueligt overblik over hvor meget man kommer til at skulle betale i procent om året af sit lån. Der er mange ting som påvirker ÅOP, om det er den givne rente eller stiftelsesgebyr eller andre afgifter varierer fra lånegiver til lånegiver.

Ekstensionel definition

Ved et lånetilbud vil man altid udregne ÅOP så kunden let kan se hvad han eller hun kommer til at skulle betale i procent om året. Det er med til at gøre det lettere for kunder at finde det rigtige lånetilbud til dem, da det ikke altid er en lav rente der et det billigste.

I vores låneaftale afdrager man med et fast beløb hver måned indtil hovedstolen er blevet betalt tilbage, der skal selvfølgelig betales renter af det resterende beløb hver måned ud over afdraget.

Konklusion (Thomas)

Vi har hele projektet igennem, forsøgt at arbejde så iterativt som muligt. Der har været steder i projektet hvor vi har afviget lidt fra den normale iterative tilgang til konstruktioner af systemer. Dette har primært været diagrammer, som ikke altid blev lavet i den rigtige rækkefølge. Dette kunne resultere i, at vi måtte sidde og rette vores diagrammer igennem mange gange, før vi kom frem til det endelige resultat.
Den iterative måde at arbejde på har dog også bragt gode ting med sig. At vi har kunne fokusere hundrede procent på og implementere 1 use case af gangen, har givet os nogle fordele, eftersom vi har arbejdet i en gruppe på fire. Det har givet os den fordel at vi kunne uddele opgaver imellem os, og være sikker på at alle havde den samme tilgangsvinkel til use casen.
Allerede tidligt i projektets forløb besluttede vi at holde os til opgavebeskrivelsen. Som i kan se i vores use case diagram, har vi use cases som ikke har noget med opgavebeskrivelsen at gøre. Men vi følte at alle vores use cases, var en del af vores ”drømme” system. Men eftersom at hver use case kræver en masse arbejde at implementere, valgte vi de mest essentielle.
Vores brugergrænseflade har vi lavet så simpel som vi synes vi kunne. Hvis vi havde flere kompetencer inden for design af brugergrænseflader, mht farver og lignende, ville vi have lavet en mere spændende brugergrænseflade.
Vi har kunne konkludere i vores projekt, at for at få en database til at opfylde 3. normalform er nemmere sagt end gjort. Især når den bliver større og større. Vi opdagede flere gange at vores database ikke levede op til 3. normalform, og måtte så igang med at rette i databasen, hvilket altid vil resultere i en masse kode der også skal rettes.
Der blev stillet et krav i opgaven omkring en CSV-fil. Dette problem gik vi forholdsvis hurtig til, for at se om det var en funktion vi kunne implementere senere i projektet uden at skulle ændre meget på systemet. Da dette ikke var tilfældet valgte vi at implementere denne funktion til sidst.
Vi kan til slut sige, at hvis vi havde haft længere tid, var systemet blevet væsentlig større, hvor alle vores use cases var blevet implementeret. Eftersom kunden ikke bad om mere, er systemet blevet et lille, men effektivt lånesystem

Litteraturliste

[i] DocJava: MVC Pattern. Udgivet af Flemming Jensen. Internetadresse:http://www.docjava.dk/patterns/model_view_controller/model_view_controller.htm - Besøgt d. 28.05.2015 (Internet)

[ii] DocJava: Singleton Pattern. Udgivet af Flemming Jensen. Internetadresse:http://www.docjava.dk/patterns/singleton/singleton.htm - Besøgt d. 28.05.2015 (Internet)

[iii] DocJava: Observer Pattern. Udgivet af Flemming Jensen. Internetadresse:http://www.docjava.dk/patterns/observer/observer.htm - Besøgt d. 28.05.2015 (Internet)

[iv] Dynamic Structure: Iterative Development. Kruchten, Phillippe : I: The Rational Unified Process An Introduction . 3. udg. Addison Wesley, 2003. Side 81-105. (Bog)

[v] Business process reengineering. Udgivet af Wikipedia. Internetadresse:http://en.wikipedia.org/wiki/Business_process_reengineering - Besøgt d. 28.05.2015 (Internet)

[vi] Other Requirements. Larman, Craig: I: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 3. udg. Addison Wesley, 2004. Side 109-114 (Bog)

[vii] Use Case Diagram. Udgivet af Wikipedia. Internetadresse: http://en.wikipedia.org/wiki/Use_Case_Diagram - Besøgt d. 28.05.2015 (Internet)

[viii] Larman, Craig : Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Side 131-171. 3. udg. Addison Wesley Professional, 2004. (Bog)

[ix] Larman, Craig : Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Side 173-180. 3. udg. Addison Wesley Professional, 2004. (Bog)

[x] Larman, Craig : Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Side 477-484. 3. udg. Addison Wesley Professional, 2004. (Bog)

[xi] Larman, Craig : Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Side 181-194. 3. udg. Addison Wesley Professional, 2004. (Bog)

[xii] Larman, Craig : Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Side 222-246. 3. udg. Addison Wesley Professional, 2004. (Bog)

[xiii] Larman, Craig : Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Side 249-270. 3. udg. Addison Wesley Professional, 2004. (Bog)

[xiv] Larman, Craig : Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Side 271-320. 3. udg. Addison Wesley Professional, 2004. (Bog)

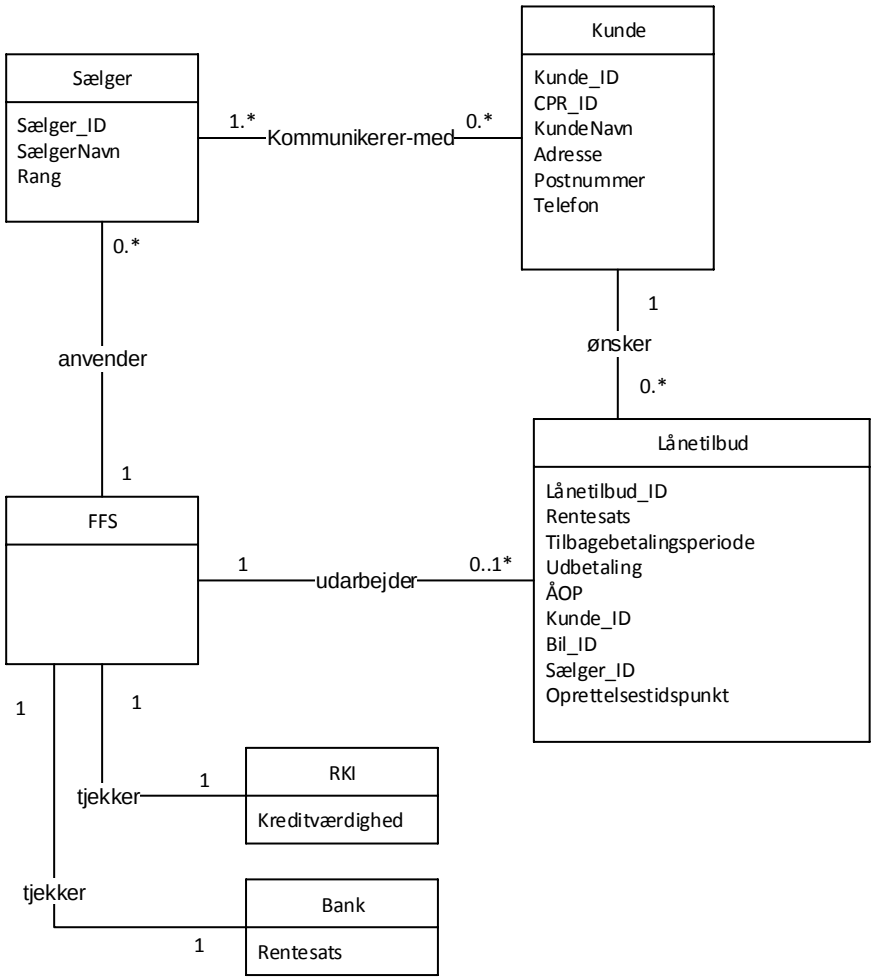
[xv] Christensen, Henrik B.: Flexible, reliable software : using patterns and agile development. Side 18-24. 1. udg. CRC Press, 2010. (Bog)

[xvi] Larman, Craig : Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Side 115. 3. udg. Addison Wesley Professional, 2004. (Bog)

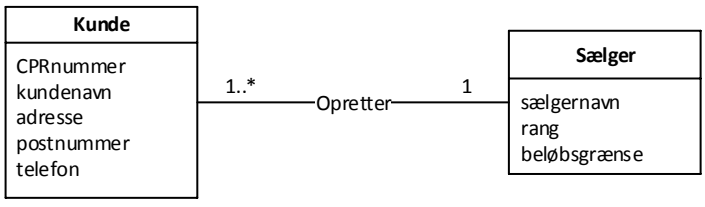
Bilag

Bilag 1

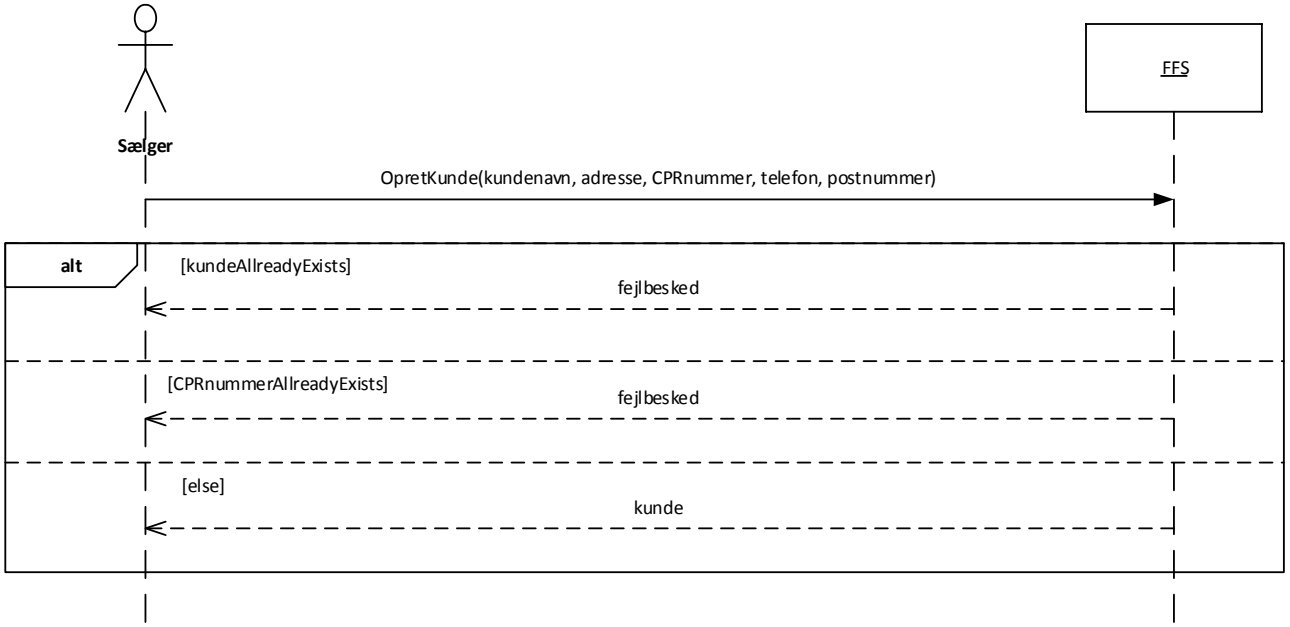
DOM: UC-7 Udarbejd Tilbud



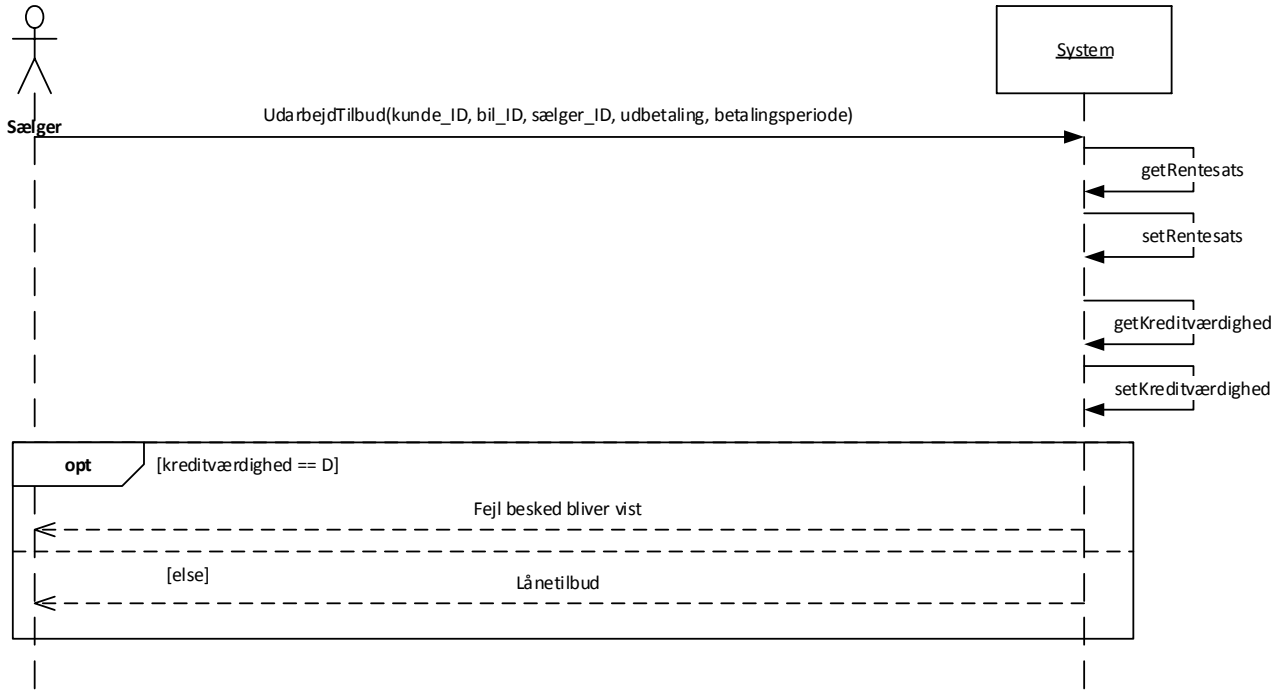
Bilag 2



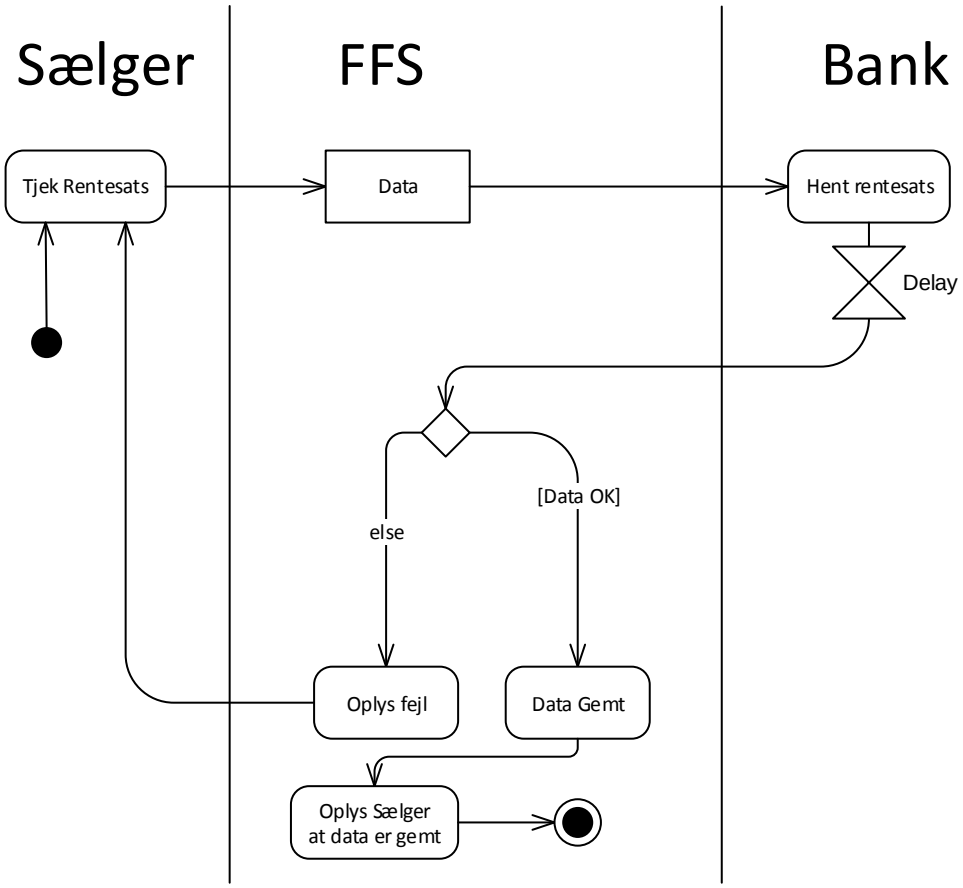
Bilag 3



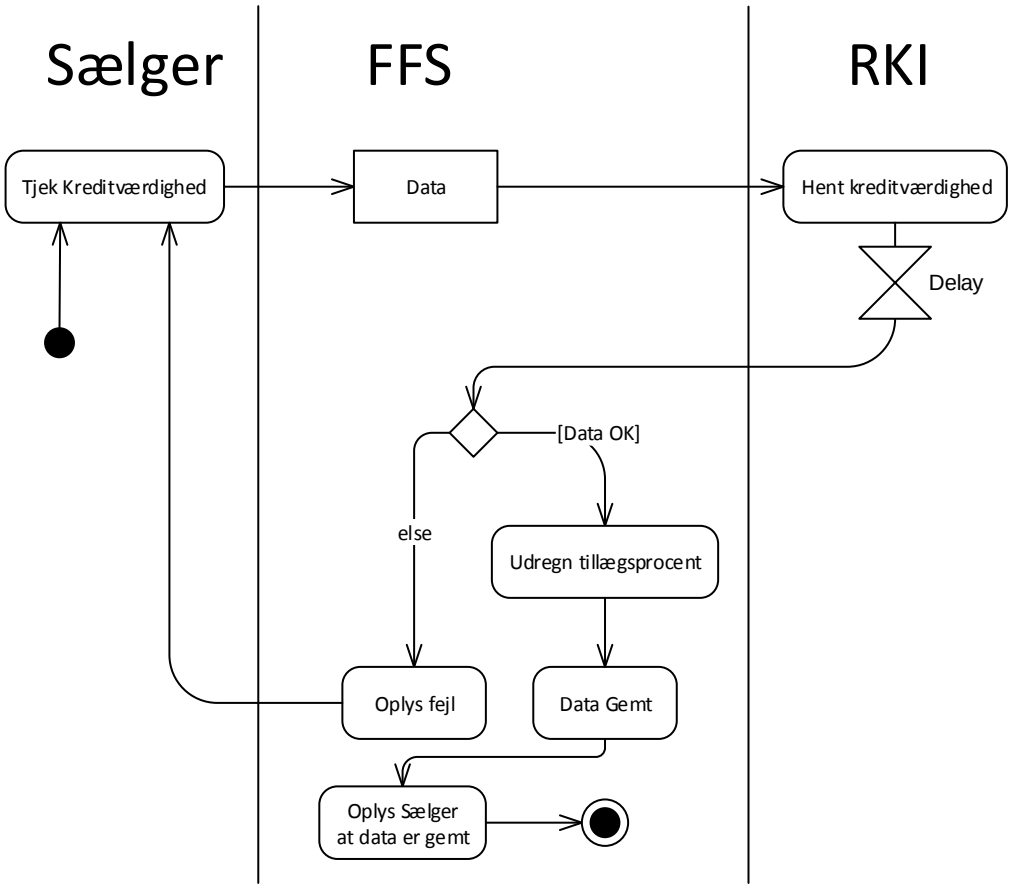
Bilag 4



Bilag 5


Sælger
FFS
Bank
Tjek Rentesats
Data
Hent rentesats
Delay
[Data OK]
else
Oplys fejl
Data Gemt
Oplys Sælger
at data er gemt

Bilag 6


Sælger
FFS
RKI
Tjek Kreditværdighed
Data
Hent kreditværdighed
Delay
[Data OK]
else
Udregn tillægsprocent
Oplys fejl
Data Gemt
Oplys Sælger
at data er gemt

Bilag 7

UC6 – OC1: setKreditværdighed

Systemoperation

setKreditværdighed(cpr : String, callback : Callback)

Krydsreferencer

Udarbejd tilbud

Forudsætninger

Forbindelse til RKI er oprettet.

Slutbetingelser

Alle instanser kvAcceptabel, tillægspoint og kreditværdighed er blevet sat.
Processen er blevet kørt i en Thread().

Bilag 8

UC5 – OC2: setRenteSats

Systemoperation

setRenteSats(callback : Callback)

Krydsreferencer

Udarbejd tilbud

Forudsætninger

Forbindelse til bank er oprettet.

Slutbetingelser

Instans rate er blevet sat til InterestRate.todaysRate().
Processen er blevet kørt i en Thread().

Bilag 9

UC2 – OC3: opretKunde

Systemoperation

opretKunde(cpr, kundenavn, adresse, postnummer, telefon)

Krydsreferencer

Udarbejd Tilbud

Forudsætninger

Ingen

Slutbetingelser

En instans kunde af Kunde er blevet skabt med tilsatte parameter (kunde_id, CPR_id, kundenavn, adresse, postnummer, telefon).

kunde.telefon blev sat til telefon.

kunde.cpr blev sat til cpr.

kunde.navn blev sat til navn.

kunde.adresse blev sat til adresse.

kunde.postnummer blev sat til postnummer.

kunde_id er blevet auto genereret af databasen.

CPR_id er blevet auto genereret af databasen.

Bilag 10

UC7 – OC4: udregnÅOP

Systemoperation

udregnÅOP(tilbageBetalingsperiode, udbetaling, rentesats, pris)

Krydsreferencer

Udarbejd Tilbud

Forudsætninger

Ingen

Slutbetingelser

En instans startgæld er blevet sat til pris – udbetaling.

En instans sum er blevet udregnet udfra udbetaling, pris og rentesats.

En instans OP(Omkostnings periode) er blevet sat til sum/startgæld.

En instans ÅOP er blevet sat til (OP/(tilbageBetalingsperiode/12))*100.

Instans af ÅOP er blevet retuneret.

Bilag 11

UC7 – OC5: beregnLånetilbud

Systemoperation

beregnLånetilbud(kunde_id)

Krydsreferencer

Udarbejd Tilbud

Tjek RenteSats

Tjek Kreditværdighed

Forudsætninger

Kunden er blevet oprettet i databasen.

Slutbetingelser

En instans kv af KreditværdighedImpl er blevet skabt.

En instans kc af KundeController er blevet skabt.

En instans kunde af er blevet sat til en kunde.

En instans cl af CPRLogikImpl er blevet skabt.

En instans cp af typen CPRnummer er blevet skabt.

En instans rs af RenteSatsImpl er blevet skabt.

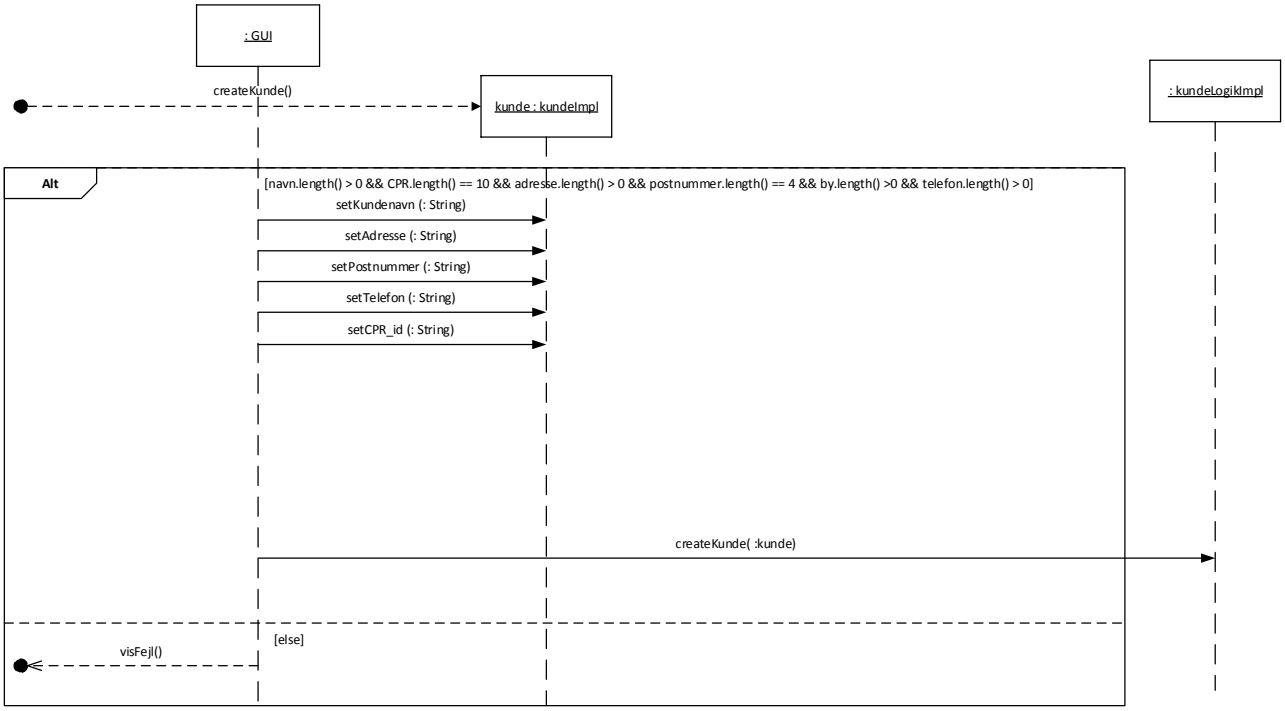
setKreditværdighed er blevet kaldt på instansen kv.

setRenteSats er blevet kaldt på på instansen rs.

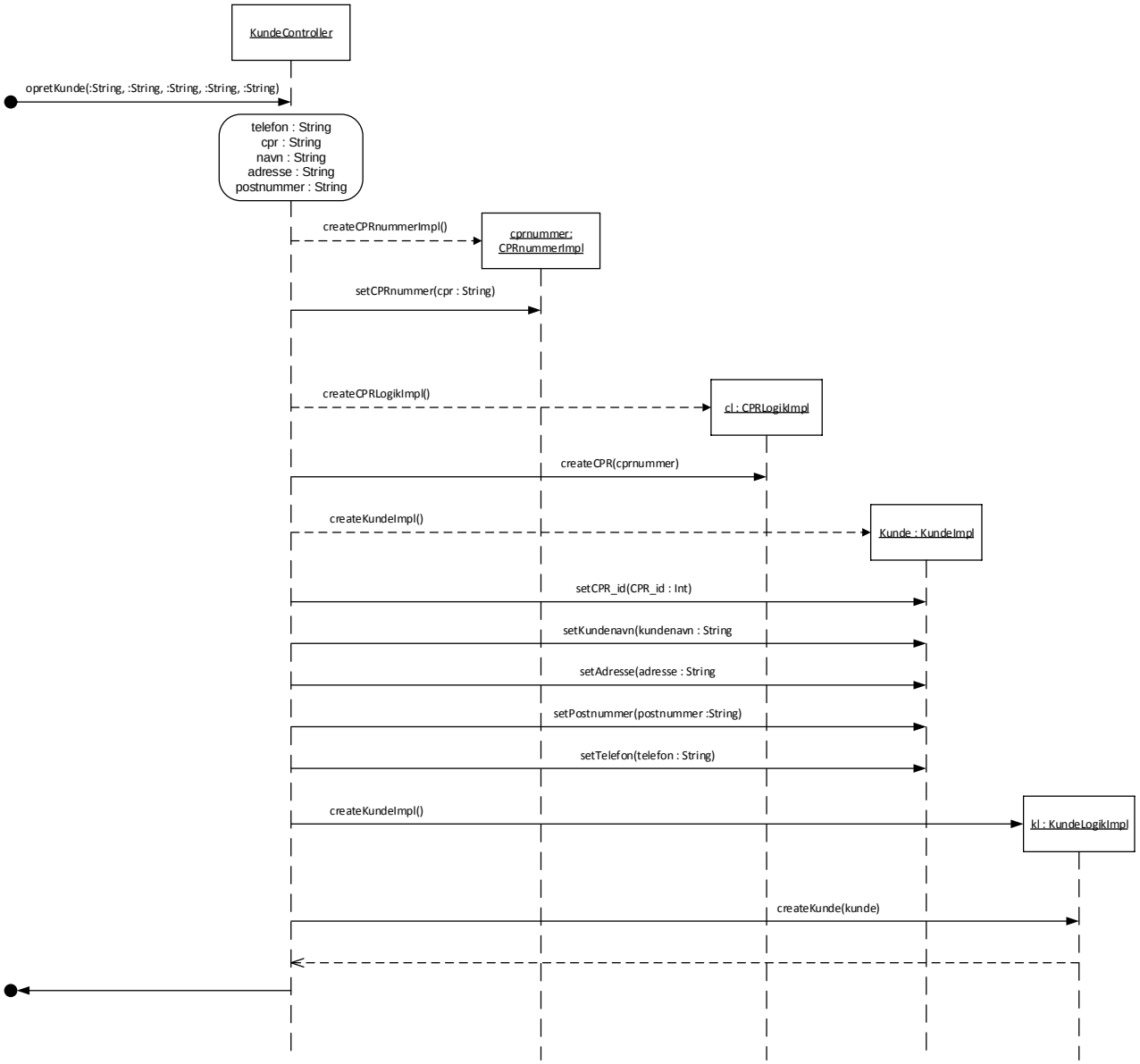
notifyOberservers er blevet kaldt.

Alle Observers har opdateret deres viabler.

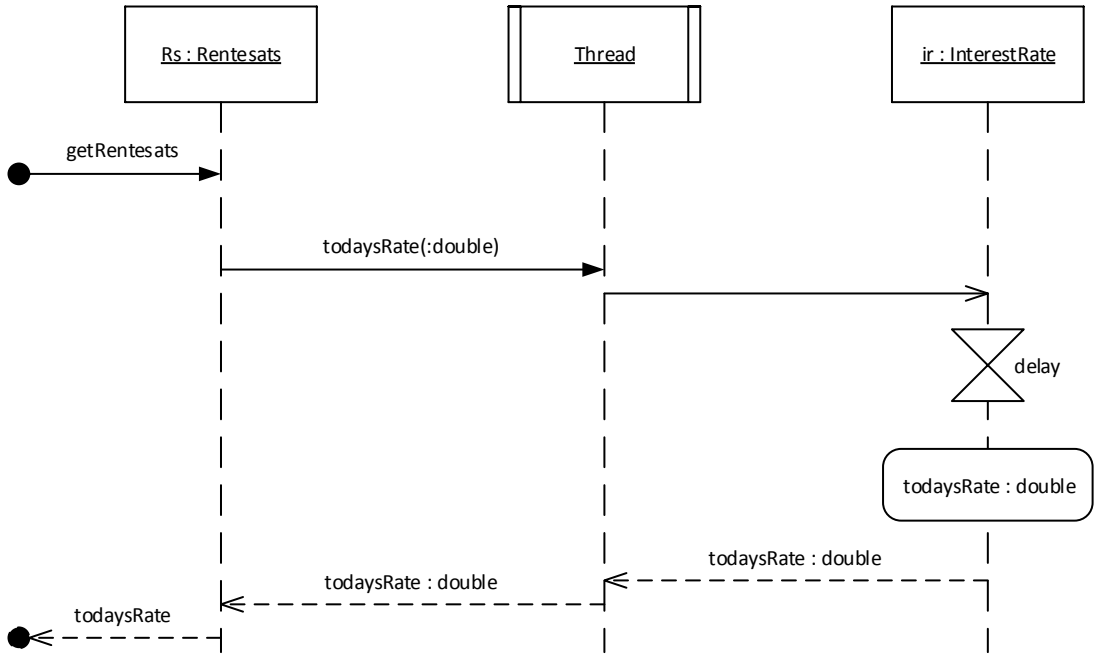
Bilag 12


: GUI
createKunde()
kunde : kundeImpl
: kundeLogikImpl
Alt
[navn.length() > 0 && CPR.length() == 10 && adresse.length() > 0 && postnummer.length() == 4 && by.length() >0 && telefon.length() > 0]
setKundenavn (: String)
setAdresse (: String)
setPostnummer (: String)
setTelefon (: String)
setCPR_id (: String)
createKunde(:kunde)
[else]
visFejl()

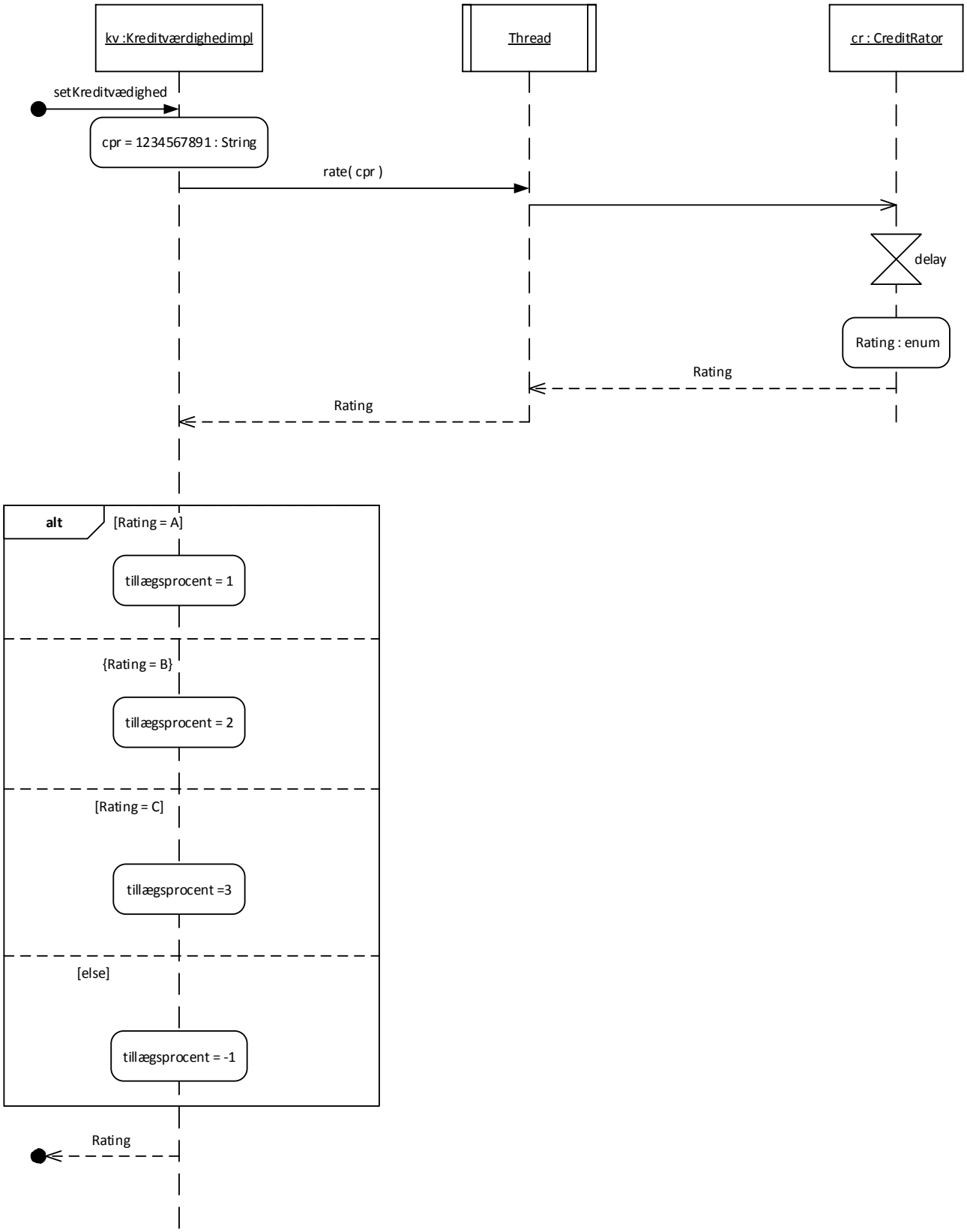
Bilag 13


KundeController
opretKunde(:String, :String, :String, :String, :String)
telefon : String
cpr : String
navn : String
adresse : String
postnummer : String
createCPRnummerImpl()
cprnummer: CPRnummerImpl
setCPRnummer(cpr : String)
createCPRLogikImpl()
cl : CPRLogikImpl
createCPR(cprnummer)
createKundeImpl()
Kunde : KundeImpl
setCPR_id(CPR_id : Int)
setKundenavn(kundenavn : String
setAdresse(adresse : String
setPostnummer(postnummer :String)
setTelefon(telefon : String)
createKundeImpl()
kl : KundeLogikImpl
createKunde(kunde)

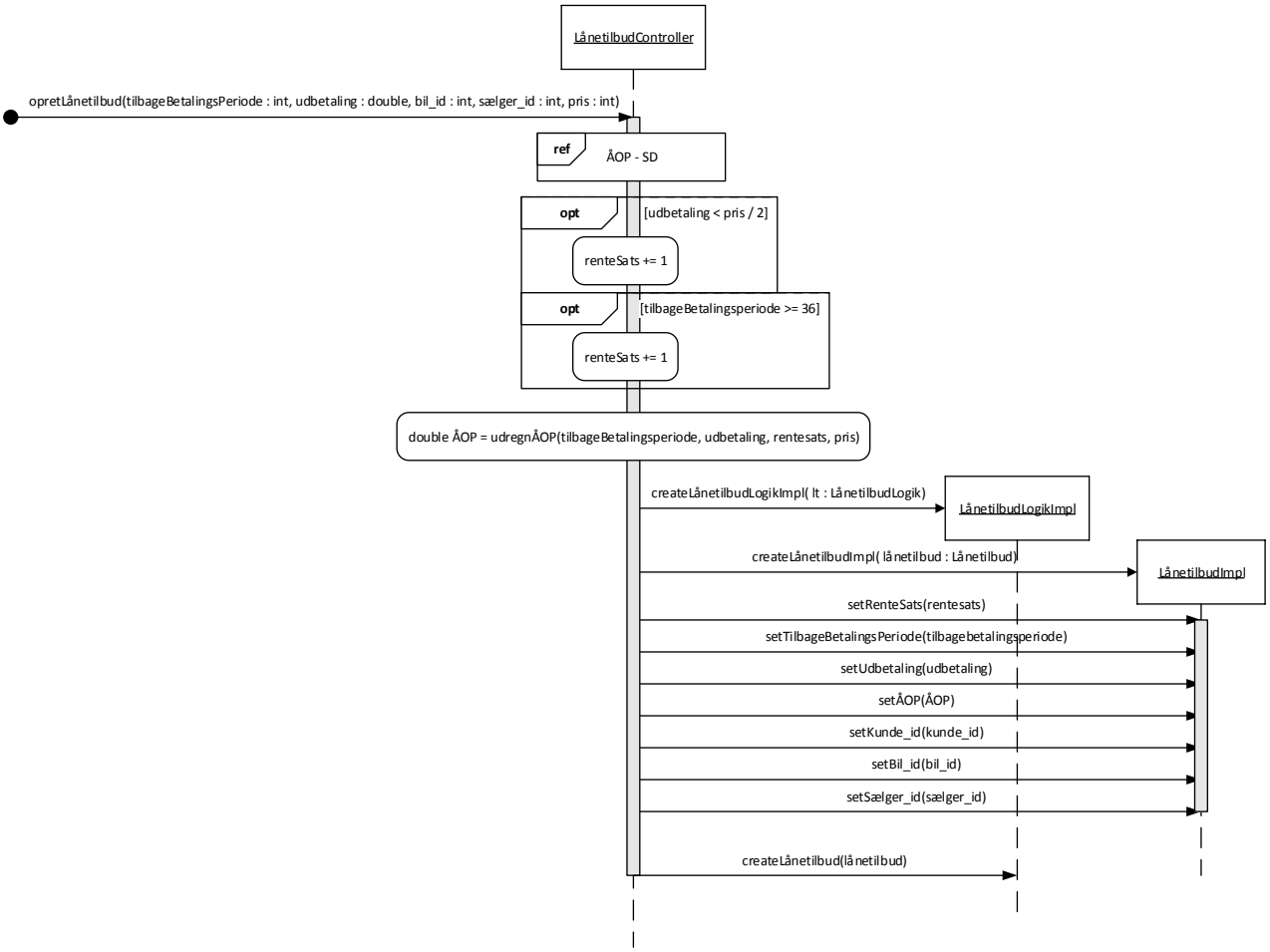
Bilag 14


Rs : Rentesats
Thread
ir : InterestRate
getRentesats
todaysRate(:double)
delay
todaysRate : double
todaysRate : double
todaysRate : double
todaysRate

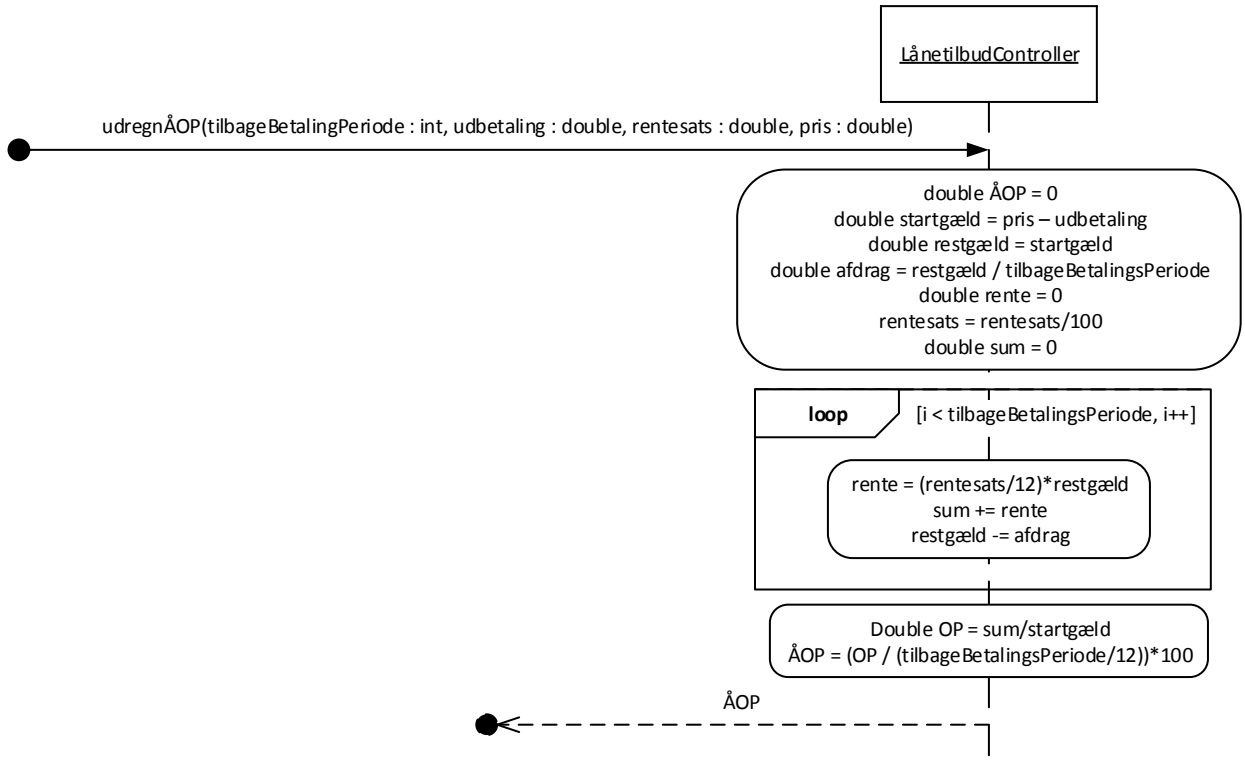
Bilag 15


kv :Kreditværdighedimpl
Thread
cr : CreditRator
setKreditvædighed
cpr = 1234567891 : String
rate(cpr)
delay
Rating : enum
Rating
Rating
alt
[Rating = A]
tillægsprocent = 1
{Rating = B}
tillægsprocent = 2
[Rating = C]
tillægsprocent =3
[else]
tillægsprocent = -1
Rating

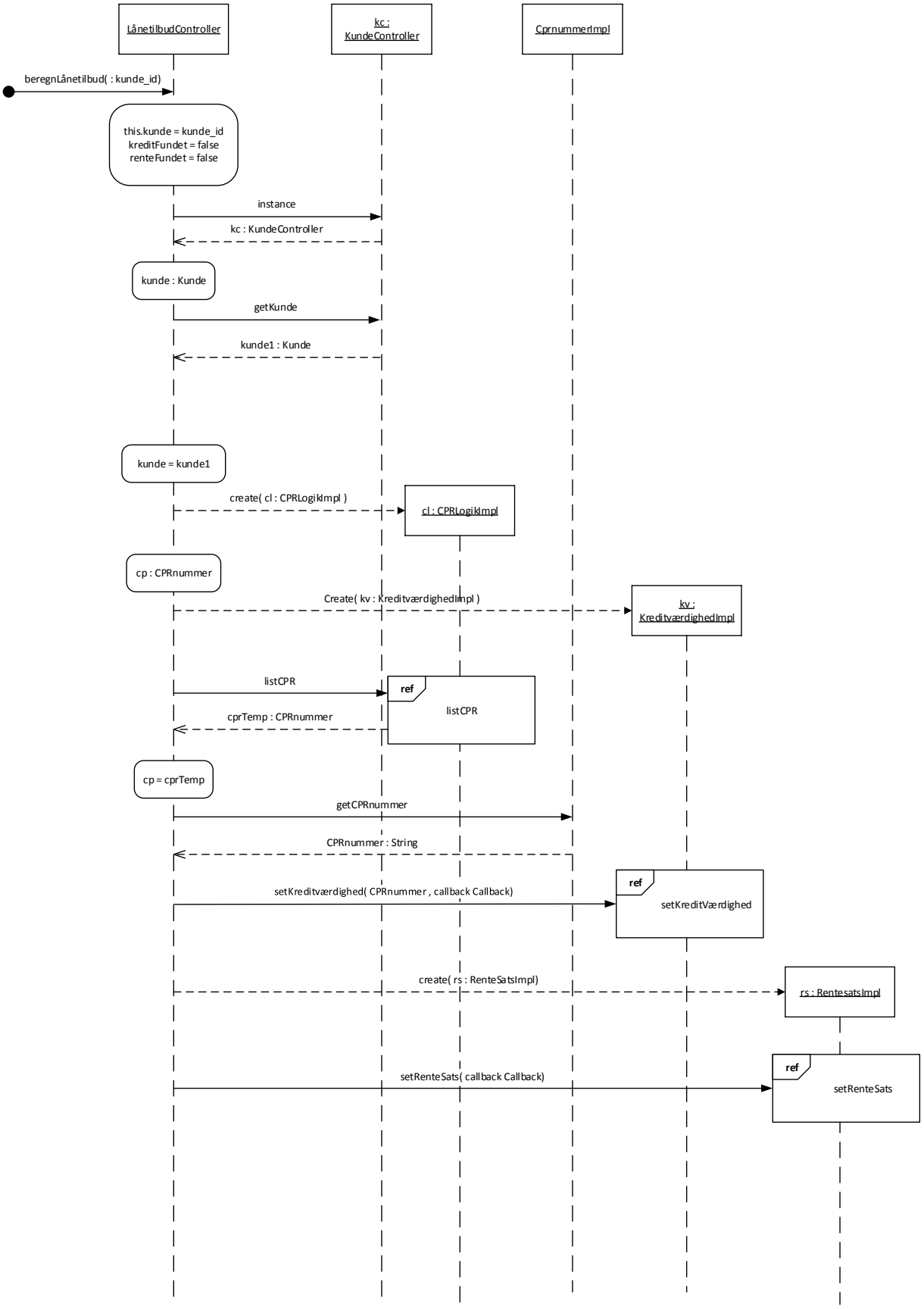
Bilag 16


LånetilbudController
opretLånetilbud(tilbageBetalingsPeriode : int, udbetaling : double, bil_id : int, sælger_id : int, pris : int)
ref
ÅOP - SD
opt
[udbetaling < pris / 2]
renteSats += 1
opt
[tilbageBetalingsperiode >= 36]
renteSats += 1
double ÅOP = udregnÅOP(tilbageBetalingsperiode, udbetaling, rentesats, pris)
createLånetilbudLogikImpl(lt : LånetilbudLogik)
LånetilbudLogikImpl
createLånetilbudImpl(lånetilbud : Lånetilbud)
LånetilbudImpl
setRenteSats(rentesats)
setTilbageBetalingsPeriode(tilbagebetalingsperiode)
setUdbetaling(udbetaling)
setÅOP(ÅOP)
setKunde_id(kunde_id)
setBil_id(bil_id)
setSælger_id(sælger_id)
createLånetilbud(lånetilbud)

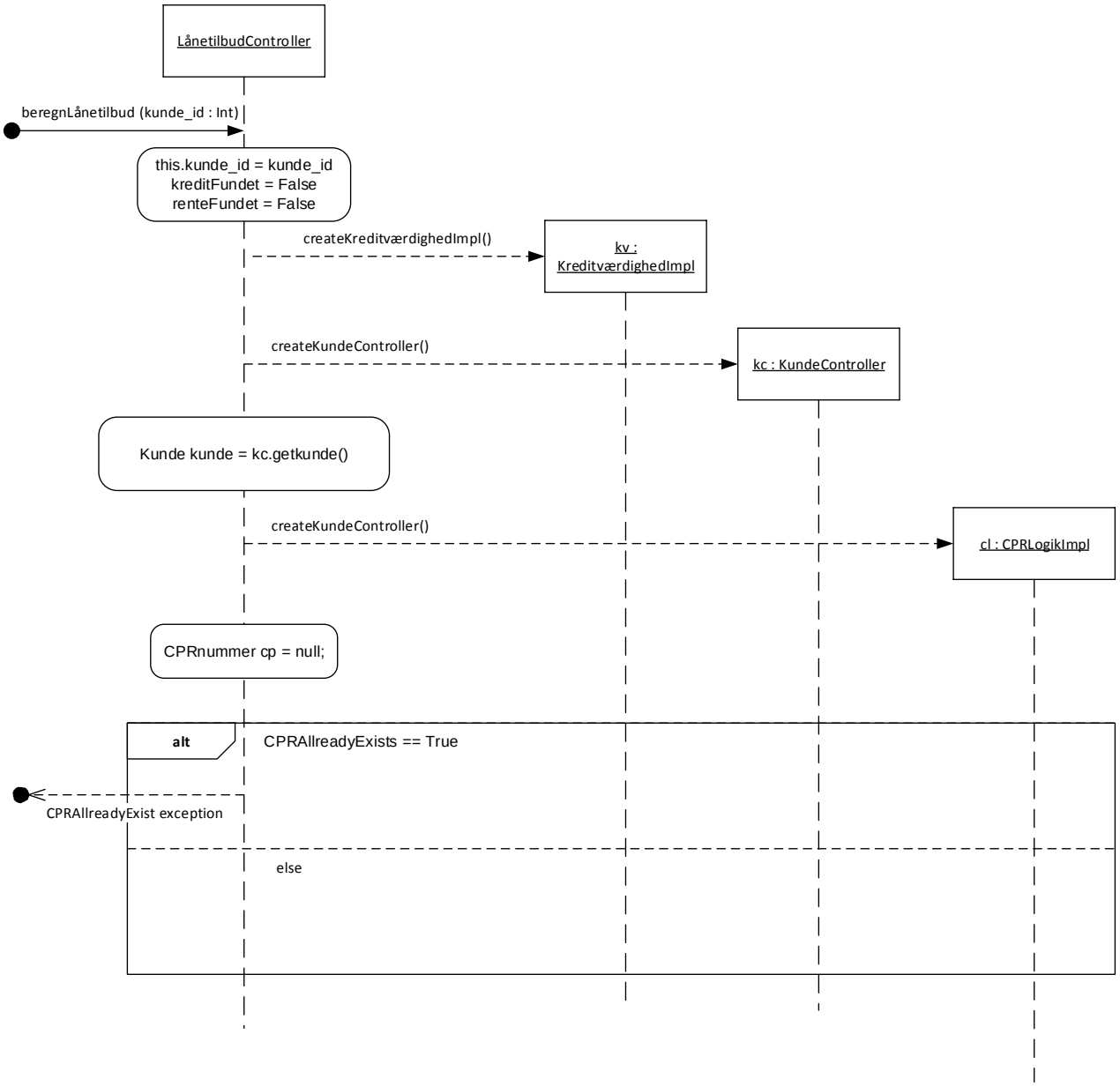
Bilag 17


LånetilbudController
udregnÅOP(tilbageBetalingPeriode : int, udbetaling : double, rentesats : double, pris : double)
double ÅOP = 0
double startgæld = pris – udbetaling
double restgæld = startgæld
double afdrag = restgæld / tilbageBetalingsPeriode
double rente = 0
rentesats = rentesats/100
double sum = 0
loop
[i < tilbageBetalingsPeriode, i++]
rente = (rentesats/12)*restgæld
sum += rente
restgæld -= afdrag
Double OP = sum/startgæld
ÅOP = (OP / (tilbageBetalingsPeriode/12))*100
ÅOP

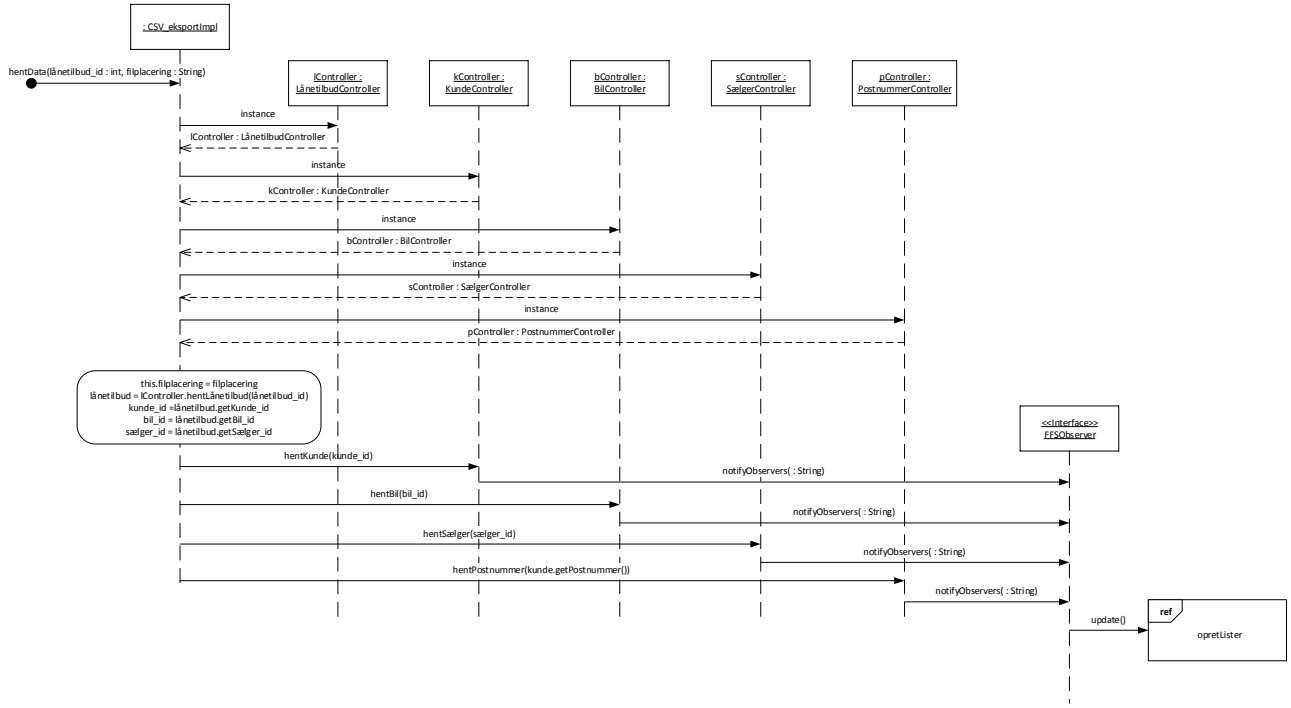
Bilag 18


LånetilbudController
kc : KundeController
CprnummerImpl
beregnLånetilbud(: kunde_id)
this.kunde = kunde_id
kreditFundet = false
renteFundet = false
instance
kc : KundeController
kunde : Kunde
getKunde
kunde1 : Kunde
kunde = kunde1
create(cl : CPRLogikImpl)
cl : CPRLogikImpl
cp : CPRnummer
Create(kv : KreditværdighedImpl)
kv : KreditværdighedImpl
listCPR
ref
listCPR
cprTemp : CPRnummer
cp = cprTemp
getCPRnummer
CPRnummer : String
setKreditværdighed(CPRnummer , callback Callback)
ref
setKreditVærdighed
create(rs : RenteSatsImpl)
rs : RentesatsImpl
ref
setRenteSats
setRenteSats(callback Callback)

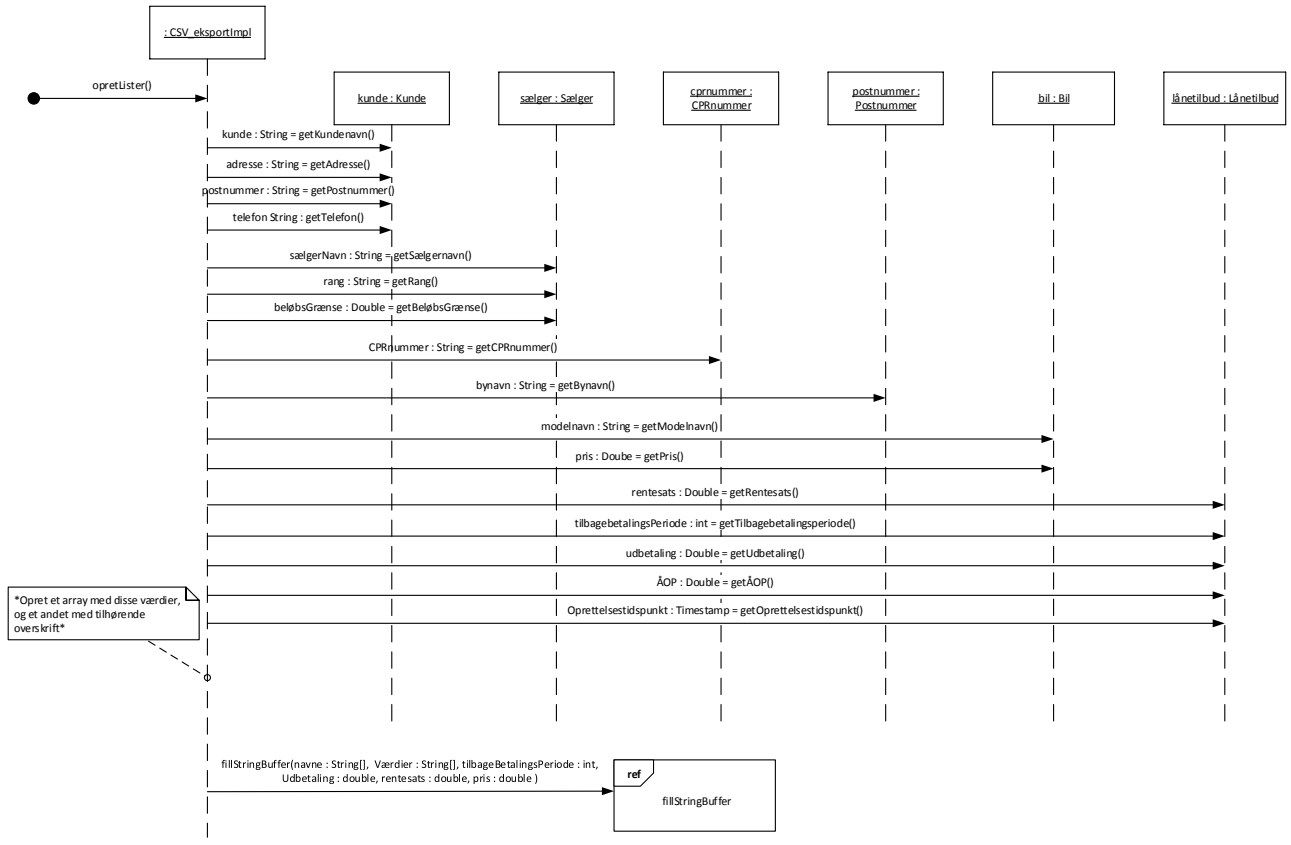
Bilag 19


LånetilbudController
beregnLånetilbud (kunde_id : Int)
this.kunde_id = kunde_id
kreditFundet = False
renteFundet = False
createKreditværdighedImpl()
kv : KreditværdighedImpl
createKundeController()
kc : KundeController
Kunde kunde = kc.getkunde()
createKundeController()
cl : CPRLogikImpl
CPRnummer cp = null;
alt
CPRAllreadyExists == True
CPRAllreadyExist exception
else

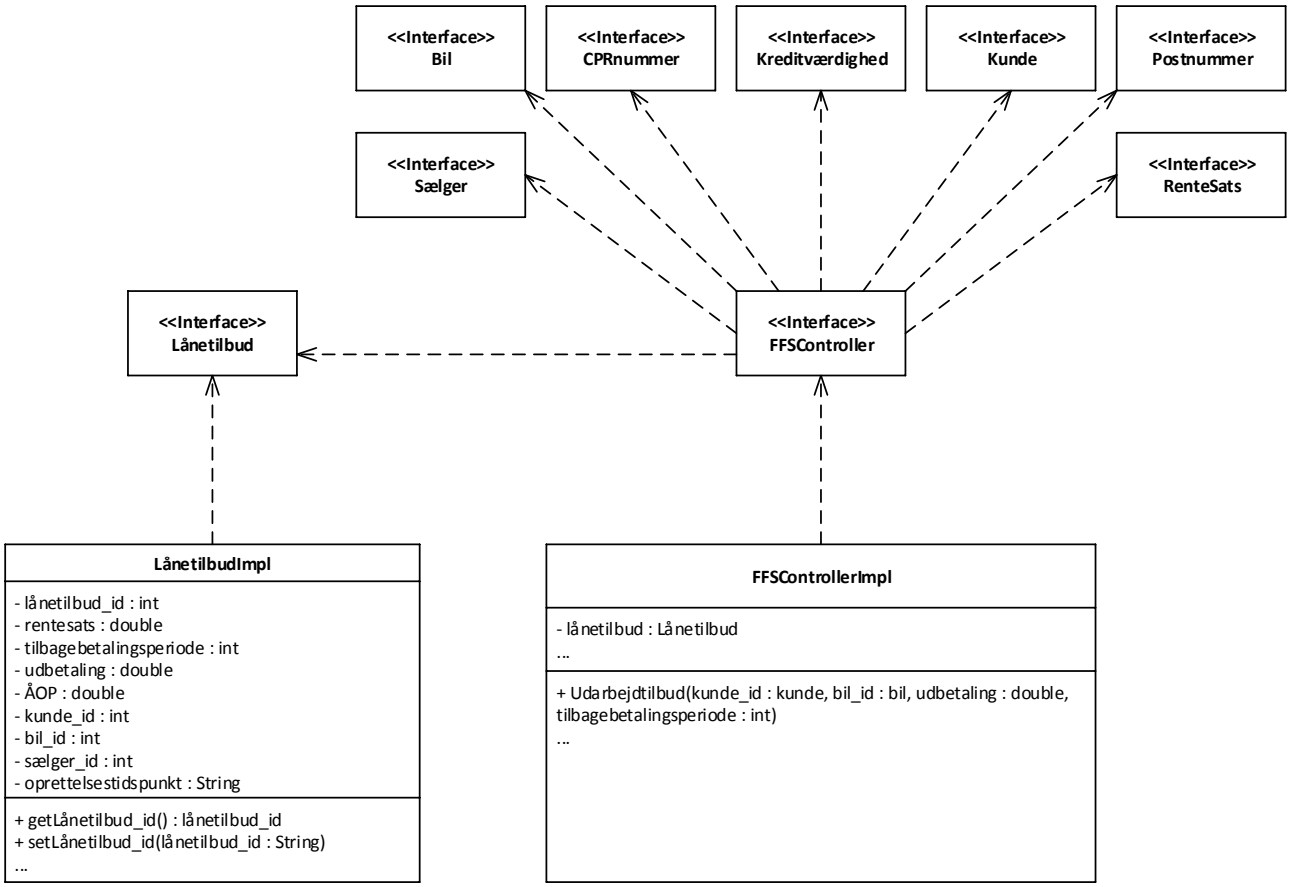
Bilag 20



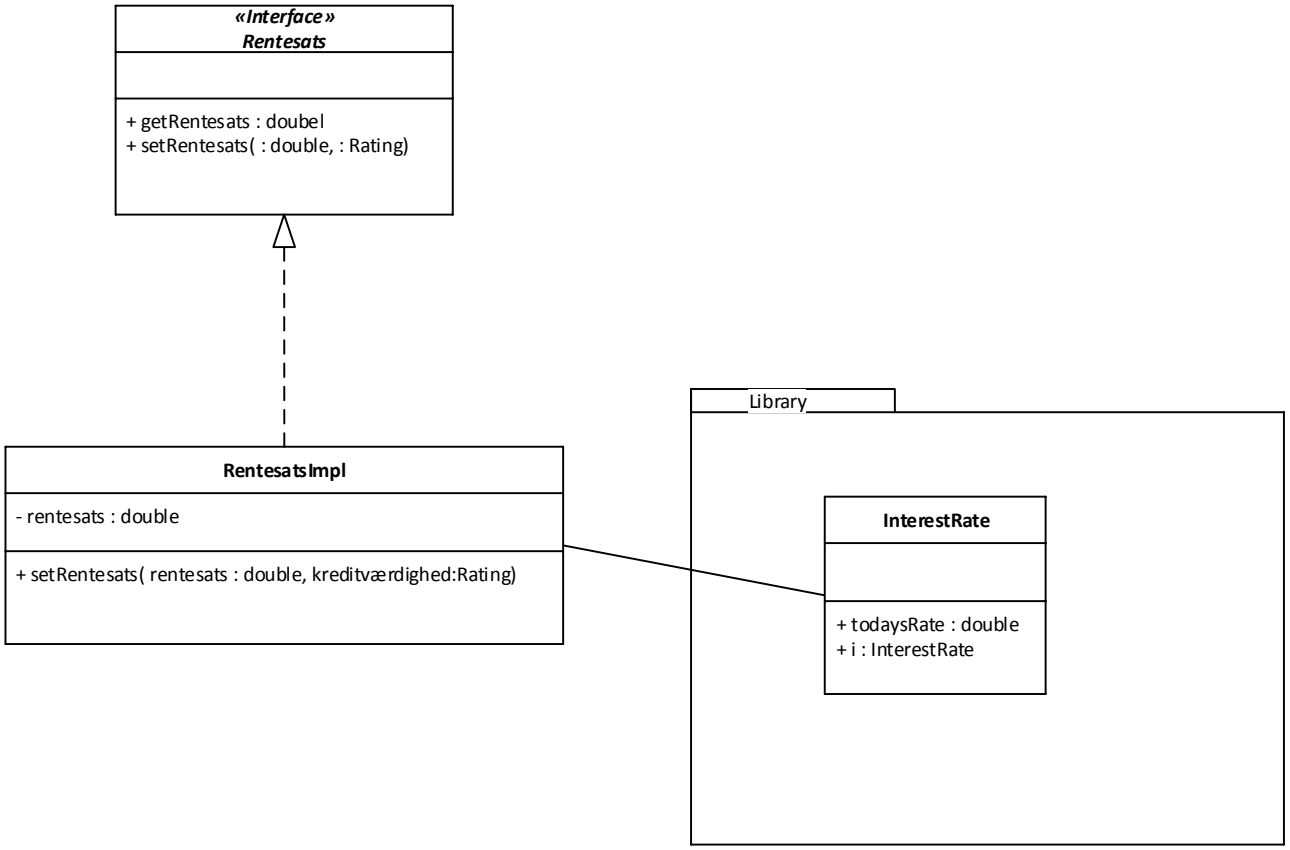
Bilag 21



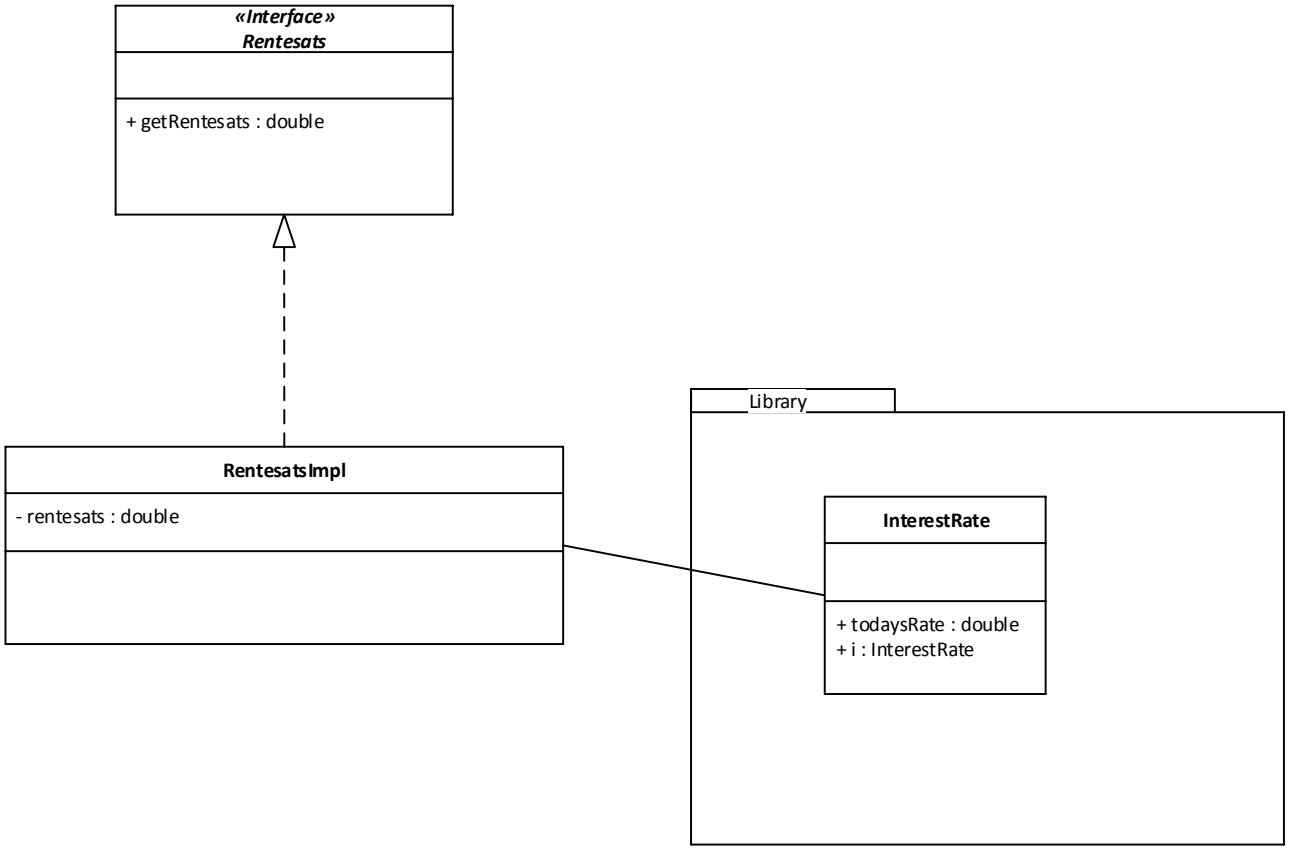
Bilag 22


<<Interface>>
Bil
<<Interface>>
CPRnummer
<<Interface>>
Kreditværdighed
<<Interface>>
Kunde
<<Interface>>
Postnummer
<<Interface>>
Sælger
<<Interface>>
RenteSats
<<Interface>>
Lånetilbud
<<Interface>>
FFSController
LånetilbudImpl
- lånetilbud_id : int
- rentesats : double
- tilbagebetalingsperiode : int
- udbetaling : double
- ÅOP : double
- kunde_id : int
- bil_id : int
- sælger_id : int
- oprettelsestidspunkt : String
+ getLånetilbud_id() : lånetilbud_id
+ setLånetilbud_id(lånetilbud_id : String)
...
FFSControllerImpl
- lånetilbud : Lånetilbud
...
+ Udarbejdtilbud(kunde_id : kunde, bil_id : bil, udbetaling : double, tilbagebetalingsperiode : int)
...

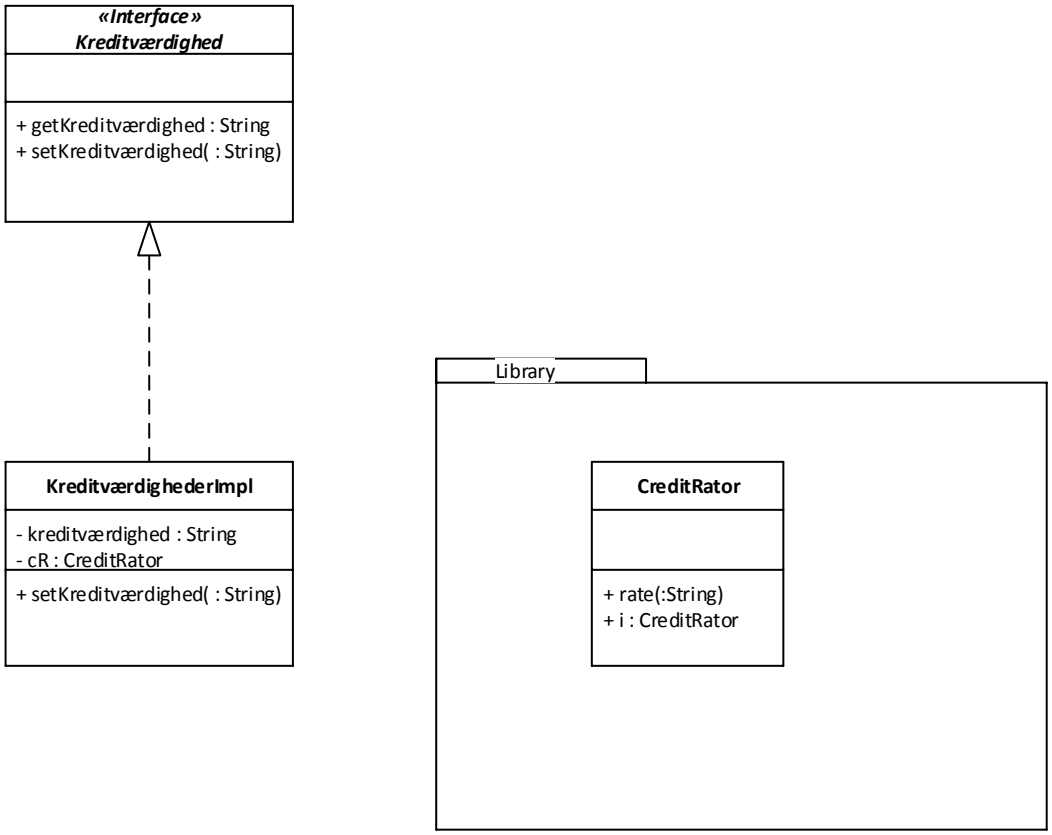
Bilag 23


«Interface»
Rentesats
+ getRentesats : doubel
+ setRentesats(: double, : Rating)
RentesatsImpl
- rentesats : double
+ setRentesats(rentesats : double, kreditværdighed:Rating)
Library
InterestRate
+ todaysRate : double
+ i : InterestRate

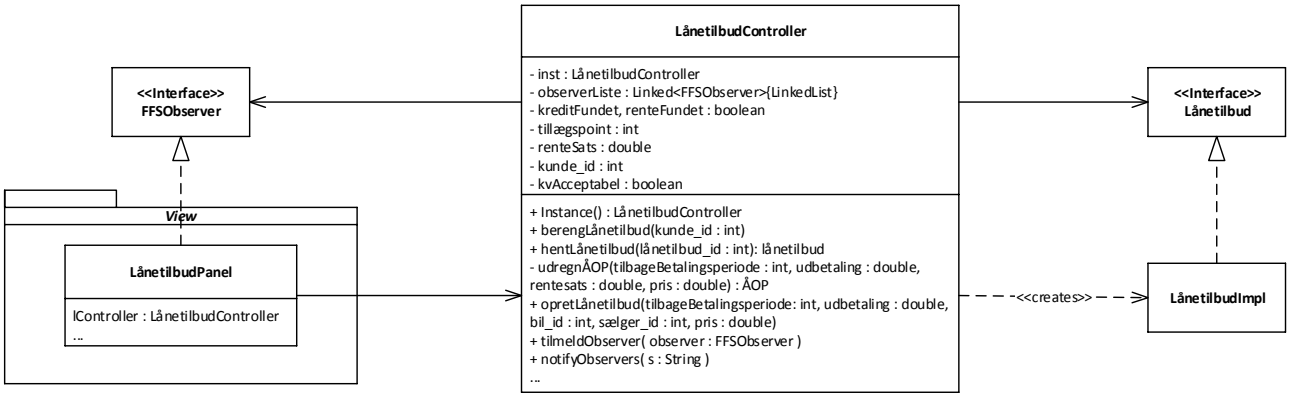
Bilag 24


«Interface»
Rentesats
+ getRentesats : double
RentesatsImpl
- rentesats : double
Library
InterestRate
+ todaysRate : double
+ i : InterestRate

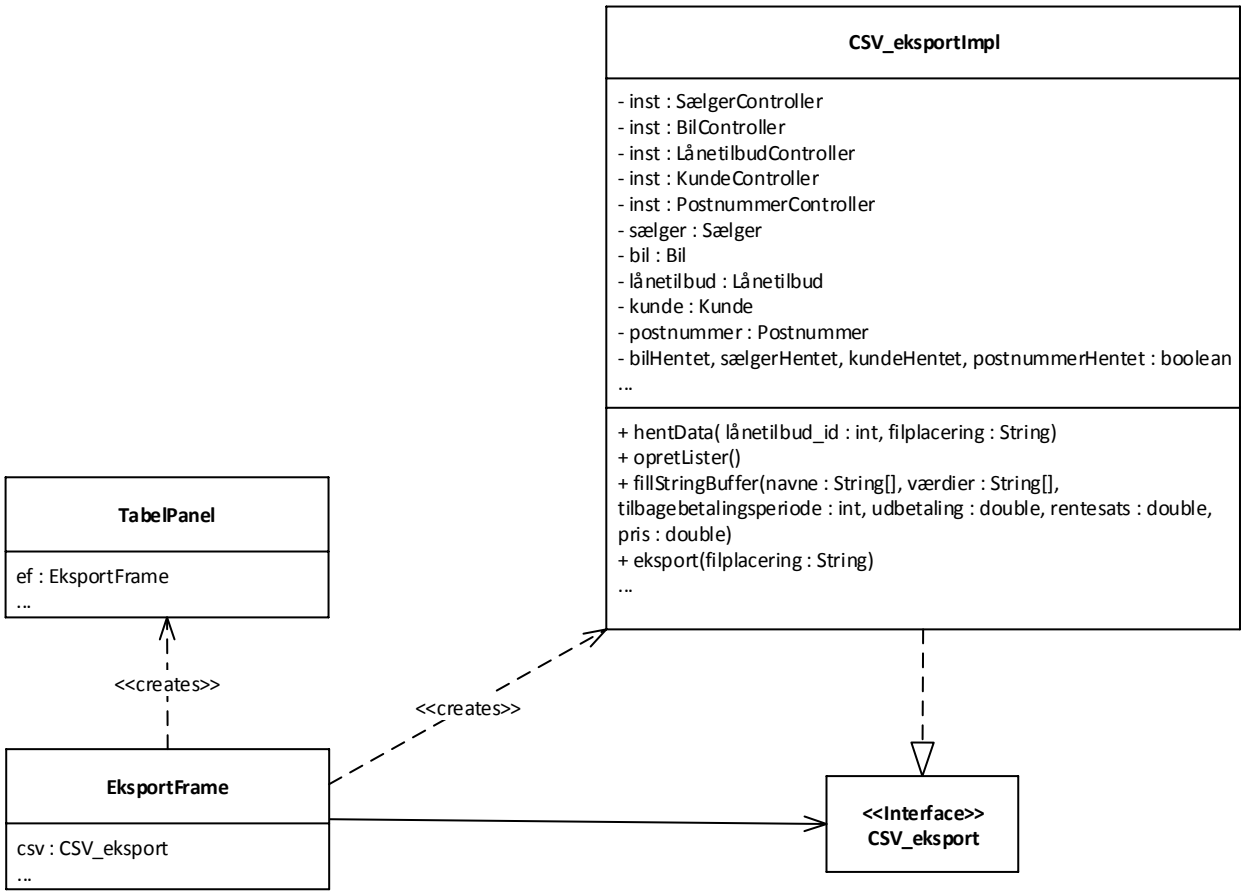
Bilag 25



Bilag 26



Bilag 27



Bilag 28

UC1 – Find kunde

Sælgeren vil gerne finde en kunde. Sælgeren angiver nogle søgekriterier for en kunde. Systemet søger i databasen over kunder. Systemet viser den søgte kunde, samt en oversigt over tideligere lån.

Hvis kunden ikke eksistere i databasen, så fortæller systemet sælgeren at, kunde ikke kan findes. Systemet går tilbage til normal tilstand.

Bilag 29

UC2 – Opret kunde

Sælger vil oprette en ny kunde i systemet. Kunde er ikke tidligere oprettet i systemet. Sælger opretter kunden i systemet med tilhørende information.

Bilag 30

UC3 – Rediger kunde

Sælgeren angiver at en given kunde skal redigeres. Sælger redigere kundens informationer og beder systemet om at gemme.

Bilag 31

FFS-6: Tjek kreditværdighed

Sælger har angivet at der skal oprettes et lån. Systemet kontakter bank, og får den givet rentesats. Systemet tjekker den ønskede tilbagebetalingsperiode. Systemet tjekker den givet udbetaling. Systemet tjekker kundens kreditværdighed. Systemet lægger bankens rentesats sammen med de andre informationer den har hentet. Systemet sætter den beregnet rentesats ind i lånetilbud.

Hvis den ønskede tilbagebetalingsperiode er over 3 år, skal der tillægges +1 procent point til bankens rentesats. Hvis udbetalingen er under 50% er bilens beløb, tillægges der +1 procent point til bankens rentesats. Hvis kundens kreditværdighed er A, bliver +1 procent point lagt til bankens rentesats. Hvis kundens kreditværdighed er B, bliver +2 procent point lagt til bankens rentesats. Hvis kundens kreditværdighed er C, bliver +3 procent point lagt til bankens rentesats.

Bilag 32

UC1
Find kunde

Scope
FFS

Level
User goal

Primary Actor
Sælger

Stakeholders and interests
Sælger – har interesse i hurtigt og nemt at kunne finde en eksisterende kunde
Salgschef – har interesse i at kundes oplysninger ikke skal oprettes igen, da dette sparer tid
Kunde – har interesse i hurtigt at kunne blive ekspederet
Datatilsynet – har interesse i at personfølsomme oplysninger opbevares korrekt, og kun vises hvor det er nødvendigt

Preconditions
Der er oprettet forbindelse til databasen, og sælgeren har telefon nummer på den kunde han/hun ønsker at finde.

Succes Guarantee
Kunde er blevet præsenteret med alle oplysninger, undtagen CPRnummer.

Main Succes Scenario
1. Sælgeren angiver den givne kundes telefonnummer
2. Sælgeren anmoder systemet om at finde kunden
3. Systemet viser alle oplysninger om kunden, undtagen CPRnummer

Extensions

2a Hvis telefonnummeret ikke er 8 tal

1. Systemet viser en fejl
2. Fortsæt fra pkt. 1

2b Hvis forbindelsen til databasen går tabt

1. Systemet viser en fejl
2. Fortsæt fra pkt. 1

3a Hvis kunden ikke eksistere I systemet

1. Fortsæt fra UC2

Special Requirements

Technology and Data Variations List

Frequency of Occurrence
60 gange I timen I peak

Bilag 33

UC2
Opret kunde

Scope
FFS

Level
User goal

Primary Actor
Sælger

Stakeholders and interests
Sælger – har interesse i hurtigt og nemt at kunne oprette en kunde
Salgschef – har interesse i at kundes oplysninger bliver oprettet korrekt
Kunde – har interesse i at hans oplysninger bliver oprettet korrekt og bliver lagret fortroligt
Datatilsynet – har interesse i at personfølsomme oplysninger opbevares korrekt

Preconditions
Der er oprettet forbindelse til databasen, og sælger har de relevante oplysninger klar

Succes Guarantee
Kunden er blevet oprettet med korrekte oplysninger

Main Succes Scenario
1. Sælger indtaster kundens telefon nr.
2. Sælger indtaster kundens cpr nr.
3. Sælger indtaster kundens navn.
4. Sælger indtaster kundens adresse.
5. Sælger indtaster kundens postnummer.
6. Systemet sætter bynavn tilhørende postnummeret.
7. Sælger trykker "opret kunde".
8. Systemet opretter kunde i databasen med tilhørende oplysninger.

Extensions
1a Hvis kundens telefon nr allerede findes i systemet

1. Systemet udfylder de resterende felter med kundens information.
2. Afslut.

2a Hvis cpr nummer indeholder ugyldige tegn eller er for langt/kort.

1. Systemet oplyser sælger om fejlen.
2. Fortsæt fra pkt. 2

3a Hvis kundens navn indeholder andet end bogstaver.

1. Systemet oplyser sælger om fejlen.

2. Fortsæt fra pkt. 3

4a Hvis kundens adresse indeholder andre tegn end tal og bogstaver.

1. Systemet oplyser sælger om fejlen.
2. Fortsæt fra pkt. 4

5a Hvis kundens postnummer indeholder andet end tal og /eller er længere eller mindre end 4 cifre

1. Systemet oplyser sælger om fejlen.
2. Fortsæt fra pkt. 5

6a Hvis postnummeret ikke kan kædes sammen med et bynavn.

1. Systemet oplyser sælger om fejlen.
2. Fortsæt fra pkt. 6

Special Requirements

Technology and Data Variations List

Frequency of Occurrence

60 gange i timen i peak.

Bilag 34

UC5
Tjek Rentesats

Scope
FFS

Level
User goal

Primary Actor
Systemet

Stakeholders and interests
Sælger – Sælger har interesse i at vide dages rentesats.
Kunde – Kunde har interesse i at rentesatsen bliver læst korrekt.
Salgschef – Salgschefen har interesse i at rentesats bliver læst i bedst mulig kvalitet.

Preconditions
Forbindelse til banken er oprettet.

Succes Guarantee
Rentesatsen er blevet aflæst korrekt, og blevet gemt.

Main Succes Scenario
1. Systemet anmoder om rentesats.
2. Systemet henter rentesatsen fra banken.
3. Systemet returnerer rentesatsen.

Extensions
2a Hvis forbindelse til banken går tabt

1. Systemet oplyser sælger om at forbindelsen til banken ikke kan oprettes.
2. Systemet afslutter processen.

Special Requirements

Technology and Data Variations List

Frequency of Occurrence
60 gange I timen I peak.

Miscellaneous

Bilag 35

UC6
Tjek Kreditværdighed

Scope
FFS

Level
User goal

Primary Actor
Systemet

Stakeholders and interests
Sælger – Sælger har interesse i at vide kundens kreditværdighed.
Kunde – Kunde har interesse i at hans/hendes kreditværdighed bliver læst korrekt.
Salgschef – Salgschefen har interesse i at kreditværdigheden bliver læst korrekt.

Preconditions
Forbindelse til RKI er oprettet.

Succes Guarantee
Kreditværdigheden på den givne kunde, er blevet aflæst korrekt, og blevet gemt.

Main Succes Scenario
1. Systemet anmoder om kreditværdighed.
2. Systemet henter kreditværdighed fra RKI.
3. Systemet returnerer kreditværdigheden.

Extensions
2a Hvis forbindelse til RKI går tabt

1. Systemet oplyser sælger om at forbindelsen til RKI ikke kan oprettes.
2. Systemet afslutter processen.

Special Requirements

Technology and Data Variations List

Frequency of Occurrence
60 gange I timen I peak.

Miscellaneous

Bilag 36

UC7
Udarbejd tilbud

Scope
FFS

Level
User goal

Primary Actor
Sælger

Stakeholders and interests
Sælger – Sælger har interesse i at øge sit salg.
Salgschef – Salgschefen er interesseret i at tilbuddet er beregnet korrekt.
Kunde – Kunden har interesse i at det går hurtigt.

Preconditions
Der er oprettet forbindelse til bank og RKI.

Succes Guarantee
Tilbuddet er blevet udregnet korrekt og er blevet præsenteret for sælger.

Main Succes Scenario
1. Sælgeren angiver det samlede lånebeløb.
2. Sælger angiver den ønskede udbetaling.
3. Sælger angiver længden af tilbagebetalingsperioden.
4. Sælger beder systemet om at udregne tilbud.
5. Systemet tjekker den aktuelle rentesats.
6. Systemet tjekker kreditværdighed.
7. Systemet udarbejder tilbud
8. Systemet gemmer lånetilbud.

Extensions

4a Hvis udbetaling er under 50%

1. Tilbud tillægges +1 procentpoint
2. Fortsæt fra hovedscenarie pkt. 5

4b Hvis tilbagebetalingsperioden overskrider 3 år

1. Tilbud tillægges +1 procentpoint.
2. Forsæt fra hovedscenarie pkt. 5

4c Hvis den angivne udbetaling og/eller tilbagebetalingsperiode er ugyldig.

1. Systemet viser en fejl.
2. Forsæt fra hovedscenarie pkt 2.

6a Hvis kundens kreditværdighed er A.

1. Tillægges +1 procentpoint.
2. Forsæt fra hovedscenarie pkt. 7

6b Hvis kundens kreditværdighed er B.

1. Tillægges +2 procentpoint.
2. Forsæt fra hovedscenarie pkt. 7

6c Hvis kundens kreditværdighed er C.

1. Tillægges +3 procentpoint.
2. Forsæt fra hovedscenarie pkt. 7

6d Hvis kundens kreditværdighed er D eller ringere.

1. Systemet oplyser Sælger at kunden har ringe kreditværdighed.
2. Systemet opretter et afvist lånetilbud.
3. Forsæt fra hovedscenarie pkt. 8

Special Requirements

Technology and Data Variations List

Frequency of Occurrence

60 gange i timen i peak.

Miscellaneous

Bilag 37

UC8
Hent lånetilbud

Scope
FFS

Level
User goal

Primary Actor
Sælger

Stakeholders and interests
Kunde – Kunden har interesse i at tilbuddet bliver hentet korrekt, da eventuelle fejl kan betyde for høj rente

Sælger – Sælgeren har interesse i at lånetilbuddet kan hentes hurtigt, da dette formindsker ekspeditionstiden

Salgschef – Salgschefen ønsker korrekt udregning af lånetilbud, så han ikke skal 'spilde' tid på at overse lånetilbud han ikke burde

Ejer – Ejeren har interesse i at få underskrevet så mange lånetilbud som muligt, så hurtigt som muligt.

Preconditions
Der er oprettet forbindelse til databasen.
Der findes allerede en kunde og et lånetilbud i databasen

Succes Guarantee
Lånetilbuddet er korrekt hentet og vist

Main Succes Scenario
1. Sælger angiver den givne kunde
2. Sælger vælger et allerede oprettet lånetilbud
3. Systemet viser det valgte lånetilbud

Extensions
3a Forbindelsen til database går tabt

1. Systemet venter oplyser brugeren om at forbindelse er gået tabt
2. Fortsæt fra pkt 1.

Special Requirements

Technology and Data Variations List

Frequency of Occurrence
60 gange I timen I peak

Bilag 38

UC9
Slet lånetilbud

Scope
FFS

Level
User goal

Primary Actor
Sælger

Stakeholders and interests
Kunde – Kunden har interesse i at tilbuddet bliver slettet.

Sælger – er interesseret i at kunne slette et lånetilbud.

Preconditions
Der findes et lånetilbud

Succes Guarantee
Lånetilbuddet er blevet slettet

Main Succes Scenario
1. Sælger vælger et lånetilbud
2. Sælger angiver at en lånetilbud skal slettes
3. Systemet giver feedback på slettet lånetilbud

Extensions
3a Forbindelsen til database går tabt

1. Systemet venter oplyser brugeren om at forbindelse er gået tabt
2. Fortsæt fra pkt 1.

Special Requirements

Technology and Data Variations List

Frequency of Occurrence
60 gange I timen I peak

Bilag 39

UC10
Eksporter lånetilbud

Scope
FFS

Level
User goal

Primary Actor
Sælger

Stakeholders and interests
Kunde – Kunden har interesse i at tilbuddet bliver hentet hurtigt, da dette mindsker ekspeditionstiden. Desuden har kunden interesse i at alle oplysninger er korrekte, så han ikke får et forkert og/eller for dyrt lån

Sælger – Sælgeren har interesse i at lånetilbuddet kan hentes hurtigt, da dette formindsker ekspeditionstiden

Ejer – Ejeren har interesse i at få underskrevet så mange lånetilbud som muligt, så hurtigt som muligt.

Preconditions
Der er oprettet forbindelse til databasen.
Der er valgt en kunde med mindst et lånetilbud

Succes Guarantee
Lånetilbuddet er korrekt hentet og eksporteret til CSV format

Main Success Scenario
1. Sælger vælger et lånetilbud for den givne kunde
2. Sælger angiver at det valgte lånetilbud skal eksporteres
3. Systemet beder om en sti hvor lånetilbuddet skal gemmes
4. Sælger angiver stien
5. Sælger godkender eksporteringen
6. Systemet giver feedback om eksporteret CSV fil

Extensions
5a Hvis dem angivne sti ikke er en sti

1. Systemet melder fejl
2. Sælger accepterer
3. Gå til pkt. 2

5b Hvis den angivne sti ikke eksisterer

1. Systemet melder fejl
2. Sælger accepterer
3. Gå til pkt. 2

5c hvis sælgeren ikke ønsker at eksportere lånetilbuddet

1. Sælger annullerer eksporteringen
2. Gå til pkt. 1

Special Requirements

Technology and Data Variations List

Frequency of Occurrence

60 gange I timen I peak

Miscellaneous

Bilag 40

Test Suite UC1 – Find Kunde

Input: Thomas Borg Nielsen	Output: Thomas Borg Nielsen
Input: Ikke eksisterende kunde	Output: Fejl besked

Vi skal også teste alle vores søgekriterier. Da de ikke er fastlagt endnu, bliver dette lagt ind senere.

Bilag 41

Test Suite UC2 – Opret Kunde

Test case 1 – Opret Kunde
Input:
Navn: Michael Bondom
CPR-Nummer: 3006921811
Adresse: Idomvej 17
Postnummer: 7500
Telefon: 55050601
Kommentar:

Output: Denne kunde skal vi kunne finde i databasen.
Søg på: Telefon nummer, Kundenavn, Adresse, Postnummer.

Test case 2 – Opret Kunde
validerCPR
Input:
10 bogstaver.
Output:
False

Input:
10 tal
Output:
True

validerNavn

Input:
Tal
Output:
False

Input:
Thomas Borg Nielsen
Output:
True

validerAdresse

Input:
+-Holstebrovej 17
Output:
False

Input:
Holstebrovej 17
Output:
True

validerPostnummer

Input:
Bogstaver
Output:
False

Input:
7400
Output:
True

Test case 3 - KrediværdighedTest

Input:

3006921611

Output:

B

Input:

3006921611

Output:

+2 procentpoint

Test case 4 – RenteSatsTest

Input:

Fake interestRate, se jars.

- 7.49

Output:

7.49

Bilag 42

Test Suite UC3 – Rediger Kunde

Input: Samme scenarie som – Test Suite UC2 – Opret Kunde.

Output: Fejl bliver vist da Telefon, CPR-Nummer og postnummer ikke kan være bogstaver.

Bilag 43

Test Suite UC4 – Slet Kunde

Input:

Opret:

Navn: Thomas Borg Nielsen

CPR-Nummer: 3006921564

Adresse: Birk Centerpark 107D

Postnummer: 7400 Herning

Telefon: 24815501

Kommentar:

Denne kunde har købt en ferrari ENZO.

Slet kunde efter oprettelse

Output på søgning: Null