Datavetenskap

Johannes Olsson

Utveckling av ett inkassohanteringsprogram

Examensarbete, C-nivå

2006:06

Utveckling av ett inkassohanteringsprogram

Johannes Olsson

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Johannes Olsson

Godkänd, 19 januari 2006

Handledare: Hannes Persson

Examinator: Stefan Lindskog

Sammanfattning

Denna uppsats beskriver arbetet med att utveckla ett inkassohanteringsprogram. Arbetet är utfört åt inkassobyrån Exactor International Debt Collection från Mariestad. Bakgrunden till projektet är att Exactor behöver en programvara för att hålla ordning på de inkassoärenden som de har. En programvara som passar deras sätt att arbeta.

I programmet finns det möjlighet för användaren att registrera ärenden som var och en innehåller en klient, en eller flera betalningsskyldiga (gäldenärer) och en eller flera fakturor. Vidare finns det möjlighet att söka och redigera de olika delarna i ett ärende.

Uppsatsen beskriver hur programmet är konstruerat och hur det fungerar. Den beskriver också grundligt hur det kommer sig att ett ärende kommer till inkassoförfarandet. Detta för att lättare få förståelse kring resterande del av uppsatsen. Vidare beskrivs databasens design samt hur information lagras i den.

Development of a debt collection management program

Abstract

This paper describes the work of developing a debt collection management program. The work was carried out as an assignment from Exactor, an international debt collection company in Mariestad. The motivation was that Exactor needed a program to manage the debt collection errands they have. The program would have to fit their way of working.

In the program, it is possible for the user to register errands that each contains a client, one or more that is liable for payment (debtors), and one or more invoices. The program also implements a feature for searching and editing the different parts of an errand.

The paper describes how the program is designed and how it works. The paper also describes the basic process from the beginning of a debt to the debt collection procedures. This description is for understanding the rest of the paper. Furthermore, the database design and how information is stored in the database is described.

Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Inledning

Detta examensarbete är utfört åt Exactor International Debt Collection i Mariestad. Exactor är ett företag som är specialiserade på att hjälpa andra företag att driva in de pengar som de har rätt till, en så kallad inkassobyrå. De har arbetat med indrivning av skulder i snart 14 år såväl nationellt som internationellt. Exactor arbetar till skillnad mot många andra inkassobyråer inte efter ett speciellt mönster, utan hanterar varje ärende unikt. Dessutom väljer de att hellre hantera några få stora ärenden framför många små. Med ett stort ärende menas ett ärende som omfattar ett stort kapital.

För att hålla i ordning på alla aktiva ärenden krävs ett bra verktyg. Idag använder Exactor en programvara som är anpassad för att hantera en större mängd ärenden där mycket av inkassohanteringen sker per automatik som följer ett visst mönster. Denna typ av arbetsgång överensstämmer inte med den som Exactor har. Detta har fått till följd att mycket av Exactors ärendehantering för tillfället sker manuellt med hjälp av t ex Word.

Jag har kommit i kontakt med Exactor genom mitt arbete som teknikansvarig hos Olsson Ekonomikonsulter i Falköping. Anledningen till detta är att vi på Olssons ansvarar för driften av deras nuvarande program. Min uppgift gentemot Exactor är att se till att programmet fungerar som det ska samt ansvara för uppdateringar av det. Under en längre tid har jag lagt märke till de problem som Exactor haft med programmet och tillsammans med dem kommit fram till detta program som redovisas i detta examensarbete. Syftet med examensarbetet är att utveckla en grund till ett nytt inkassohanteringssystem. Ett system som till skillnad mot deras nuvarande är anpassat efter Exactors sätt att arbeta. Målet är sedan att Exactor ska kunna använda det för att effektivisera sin ärendehantering.

1.1 Programmets funktionalitet

Eftersom tiden för examensarbetet är begränsad kommer endast en utvald del av programmet att skapas. Tillsammans med uppdragsgivaren bestämdes att examensarbetet ska omfatta utveckling av ett inkassohanteringssystem där användaren kan göra följande:

- Registrera ett nytt ärende med de delar som ett ärende innefattar såsom klient, gäldenär och faktura.
- Registrera en ny klient

- Registrera en ny faktura till ett befintligt ärende
- Ta bort en faktura från ett befintligt ärende
- Registrera en gäldenär till ett befintligt ärende
- Ta bort en gäldenär från ett befintligt ärende
- Söka efter ärende, klient, gäldenär
- Visa ett befintligt ärende
- Visa en befintlig klient
- Visa en befintlig gäldenär
- Redigera en befintlig klient
- Redigera en befintlig gäldenär

Programmet skulle ha ett grafiskt användargränssnitt som är lätt att överskåda och använda. Det fanns alltså en stor frihet vid designen av gränssnittet. Det fanns heller inga krav kring vilka verktyg som skulle användas till projektet.

1.2 Uppsatsens disposition

Uppsatsen är uppdelad i sex stycken huvudkapitel. För att lättare förstå resterande delen av uppsatsen finns kapitel 2 som bland annat beskriver vad inkasso är och hur det kommer sig att folk drabbas av inkasso. Kapitlet förklarar också några av de olika termer som används i inkassobranschen.

Programmet är skrivet i programspråket Java och använder sig av databashanteraren SQLite för att lagra information. Kapitel 3 behandlar vad SQLite och Java är samt förklarar vilka fördelar respektive nackdelar de har som varit avgörande för valet av verktyg till examensarbetet.

I kapitel 4 beskrivs databasen design. Detta genom att ingående beskriva de olika tabeller och fält som databasen är uppbyggd av. Kapitlet ger läsaren förståelse kring hur information lagras i databasen.

Kapitel 5 behandlar inkassohanteringssystemets uppbyggnad och beskriver ingående hur de olika delarna i programmet fungerar. Dessutom beskrivs hur de olika delarna i programmet samarbetar med varandra.

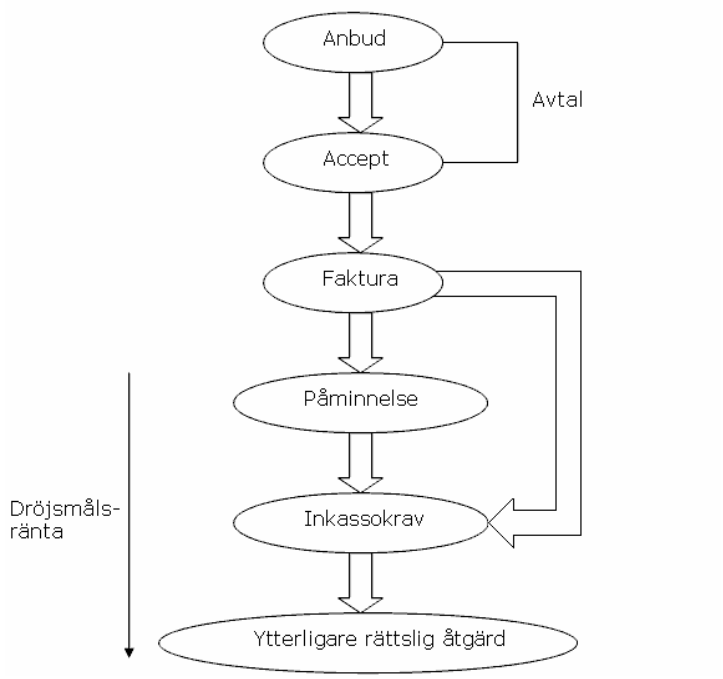
Uppsatsen avslutas med slutsatser som behandlar hur examensarbetet har gått, vilka problem som uppstått under tiden och hur arbetet med inkassohanteringssystemet ska fortgå efter examensarbetet.

2 Bakgrund

För att få en förståelse kring hur inkassohantering fungerar krävs en del bakgrundsfakta. I detta kapitel behandlas hur det kommer sig att ett ärende kommer till inkassoförfarandet. Det går också att läsa om vem som får sköta inkassoförfarandet och vad en juridisk/fysisk person är. Till sist behandlas betalningsskyldighet vid de två vanligaste bolagsformerna, aktiebolag och handelsbolag.

2.1 Från avtal till inkasso

I avsnitt 2.1.1 till och med 2.1.5 behandlas de olika faser som sker från att ett avtal slut fram till att en inkassoåtgärd har påbörjats. Hur detta går till visar Figur 2.1 översiktligt.



Figur 2-1: Flödesschema från att ett avtal sluts fram till ytterligare rättslig åtgärd.

2.1.1 Avtal

Anledningen till att en person har blivit betalningsskyldig är oftast att den slutit ett avtal med någon. Ett avtal är en överenskommelse mellan två eller flera parter som ska utföra någon typ av tjänst gentemot varandra. Avtalet kan vara muntligt såväl som skriftligt eftersom man inte ser någon skillnad på muntliga och skriftliga avtal när man syftar på hur giltigt ett avtal är.

Vissa få undantagsfall kräver dock ett skriftligt avtal. Detta är inget som kommer behandlas vidare i denna uppsats.[6]

Ett exempel på ett avtal är när en säljare skickar en offert till en köpare och köparen bestämmer sig för att godkänna offerten. I detta fall kallas offerten för avtalets anbud och köparens beslut för avtalet accept. Säljaren som lämnar anbudet är bunden till det under en viss tid. Denna tid kallas för anbudets acceptfrist och är om inget annat angivits den tid det tar att skicka anbudet plus lite betänketid [5].

2.1.2 Faktura

Vid försäljning av varor eller tjänster är det normalt att skicka en faktura till kunden. När försäljningen sker till en så kallad juridisk person (se avsnitt 2.3) krävs det att en faktura skickas. Detta är dock inget krav vid försäljning till privatpersoner. En faktura innehåller information om avtalet som slutits mellan köparen och säljaren. Man ska dock inte glömma att det är avtalet som är anledningen till att köparen är betalningsskyldig, inte fakturan.[3]

En faktura ska bland annat innehålla följande information (enligt [4]):

- Fakturadatum när fakturan skrevs ut.
- Ett fakturanummer som unikt identifierar fakturan.
- Namn och adress på köparen respektive säljaren.
- Information om vad fakturan omfattar.
- Uppgift om vilken momssats som gäller.
- Vilken moms som köparen ska betala.

2.1.3 Påminnelse

Om kunden inte betalar sin skuld i tid brukar man skicka ut en påminnelse. Påminnelsen har till uppgift att påminna kunden om att den är betalningsskyldig för att på så sätt få den att betala. När en påminnelse skickas har säljaren rätt att ta ut en avgift på maximalt 50 kr. Detta måste dock vara klartgjort i avtalet med kunden. Om kunden trots påminnelsen inte betalar sin skuld kan säljaren välja att vidta inkassoåtgärder.[3]

Det finns inga regler som säger att en påminnelse måste skickas bortsett från några få undantagsfall. Dessa är vid obetald felparkeringsavgift eller tv-avgift. Det finns heller inga krav på hur en påminnelse måste se ut och vilka uppgifter som ska framgå. Det finns dock vissa rekommendationer, dessa är (enligt [6]):

- Påminnelsen ska vara lätt att förstå
- Hur mycket som ska betalas, vad som ska betalas och vart det ska betalas
- Vem som är referenspersonen hos kunden
- När betalningskontrollen gjordes
- Vilket förfallodatum påminnelsen har
- Vilken påminnelseavgift som ska betalas ifall sådan är avtalad
- Vilken dröjsmålsränta som gäller

2.1.4 Inkasso

Inkasso är ytterligare en åtgärd för att få kunden att betala sin skuld till säljaren. När man pratar inkassotermer brukar den som ska få betalt kallas borgenär medan den som är betalningsskyldig kallas gäldenär. Detta är de termer som kommer att användas framöver i uppsatsen. Eftersom examensarbetet är utfört åt en inkassobyrå och det är deras klienter som vill ha hjälp att driva in skulder benämns ett ärendes borgenär också som klient längre fram i uppsatsen.[6]

Ett inkassoärende består av en borgenär, en eller flera gäldenärer samt de skulder som gäldenären har till borgenären. Utöver detta tillkommer det också eventuella inkassoavgifter. [3]

Den första åtgärden i ett inkassoärende är att skicka ett så kallat inkassokrav till gäldenären. Ett inkassokrav är ett skriftligt hot mot gäldenären för att få den att betala sin skuld När borgenären skickar ett inkassokrav har den rätt att ta ut en avgift på maximalt 160 kr. Inkassokravet ska vara i skriftlig form och måste enligt den inkassolag som finns innehålla följande information:

- Borgenärens namn
- Vad fordran grundar sig på
- Uppgift om kostnaden för inkassokravet
- Skuldbeloppet
- Räntan, räntesatsen samt hur lång tid ränteberäkningen grundar sig på
- Hur lång tid gäldenären har på sig att betala

Om gäldenären trots inkassokravet inte betalar sin skuld kan borgenären vidta ytterligare rättsliga åtgärder. Detta är inget som kommer behandlas vidare i denna uppsats.[2]

2.1.5 Dröjsmålsränta

När inte kunden har betalat sin skuld i rätt tid brukar den ofta drabbas av att betala dröjsmålsränta. Den ränta som ska betalas beror på det avtal som säljaren och kunden slutit. Om det i avtalet inte är sagt något angående dröjsmålsränta, gäller räntelagens regler. Räntelagen säger att säljaren har rätt att ta ut en ränta 30 dagar efter att fakturan är skickad. Räntan beräknas över ett helt år med en räntesats på 8 % plus Riksbankens gällande referensränta.[1]

Dröjsmålsräntan beräknas enligt följande formel [6]:

$$\frac{Kapitalbeloppet \times (Riksbankens\ referensränta + 8\,\%) \times antal\ förseningsdagar}{360}$$

Ett räkneexempel: Säljare A har skickat en faktura på 10000 kr med fakturadatum 2005-04-30 till kund B. Dagens datum är 2005-06-30 och det är alltså 61 dagar sedan fakturan skickades. Eftersom inget är sagt angående dröjsmålsränta i avtalet gäller räntelagens regler på 8,0 %. Det gäller också att dröjsmålsränta kan räknas från 30 dagar efter fakturadatum. Detta innebär att antalet förseningsdagar är 31. Riksbankens referensränta är vid detta tillfälle 2,0 %. Den sammanlagda räntesatsen blir alltså 10 %. Beräkningen av dröjsmålsräntan blir enligt följande:

$$\frac{10000 \times 10\,\% \times 31}{360} \approx 86\ kr$$

Borgenären har alltså rätt att denna dag ta ut en dröjsmålsränta på 86 kr.

2.2 Vem sköter indrivningen?

Säljaren kan välja om den själv vill sköta inkassohanteringen eller låta någon annan göra det, t ex en inkassobyrå. En inkassobyrå är ett företag som arbetar med att hjälpa andra med att driva in skulder. Det finns både för- och nackdelar med att sköta inkassohanteringen själv. Några fördelar är:

- Man får extra intäkter vid lyckade indrivningar i form av olika avgifter. Ett exempel är inkassokravsavgiften på maximalt 160 kr som behandlades i avsnitt 2.1.4.
- Bättre kontroll då risken för fel i informationsöverföringen till inkassobyrån inte finns.

En nackdel med att sköta inkassoförfarandet själv är:

- Extra tidsåtgång då det oftast tar mer tid att sköta inkassohanteringen själv gentemot att låta en inkassobyrå göra det.
- Företag som är inkassobyråer har en mer komplicerad lagstiftning kring inkasso.

Inkassoverksamhet ska bedrivas enligt god inkassosed. Detta innebär att gäldenären inte ska utsättas för onödiga kostnader eller trakasserier. Datainspektionen är en statlig förvaltning som bland annat har till uppgift att detta efterföljs.[1]

2.3 Fysisk / Juridisk person

I ekonomiska sammanhang pratar man ofta om fysiska och juridiska personer. En fysisk person är en människa medan en juridisk person är en sammanslutning av fysiska personer. Detta kan t ex vara ett företag eller en förening. En juridisk person kan precis som en fysisk äga tillgångar och ha skulder. De juridiska personerna som är vanligast är aktiebolag och handelsbolag. När man pratar om hur betalningskrav kan riktas mot olika typer av bolag är den viktigaste skillnaden hur ansvaret för bolagets skulder fördelas. I avsnitt 2.3.1 och 2.3.2 kommer betalningsskyldighet vid ett aktiebolag respektive ett handelsbolag behandlas lite närmare.[6]

2.3.1 Aktiebolag

Delägarna i ett aktiebolag har i normalfallet inget personligt betalningsansvar gentemot bolaget. För detta är de skyldiga att redovisa vissa uppgifter om företaget. Detta är bland annat uppgifter om företagets resultat och vilka personer som ingår i styrelsen. Det finns dock fall där vd och de övriga ledamöterna som sitter i styrelsen kan bli personligt betalningsansvariga. Detta är inget som kommer behandlas vidare i denna uppsats.[3]

2.3.2 Handelsbolag

Till skillnad mot ett aktiebolag har samtliga bolagsmän i ett handelsbolag betalningsansvar gentemot bolagets skulder. Detta gäller både skulder som fanns innan och som uppkommit under tiden man varit delägare i bolaget. Vid utträde ur ett handelsbolag blir en delägare ändå inte kvitt sitt betalningsansvar mot de skulder som den hade betalningsansvar för under sin tid i bolaget.[3]

3 Verktyg

I projektet användes programspråket Java för att skriva programmet och databasmotorn SQLite för att lagra informationen. I detta kapitel behandlas vad Java och SQLite är. Det går också att läsa vilka fördelar respektive nackdelar som dessa verktyg för med sig. Den största anledningen till att Java och SQLite blev lämpliga verktyg för projektet var att jag kommit i kontakt med dem tidigare. En annan anledning är att de båda är gratis att använda. Slutligen behandlas hur dessa två verktyg kopplas samman med hjälp JDBC, detta i avsnitt 3.3.

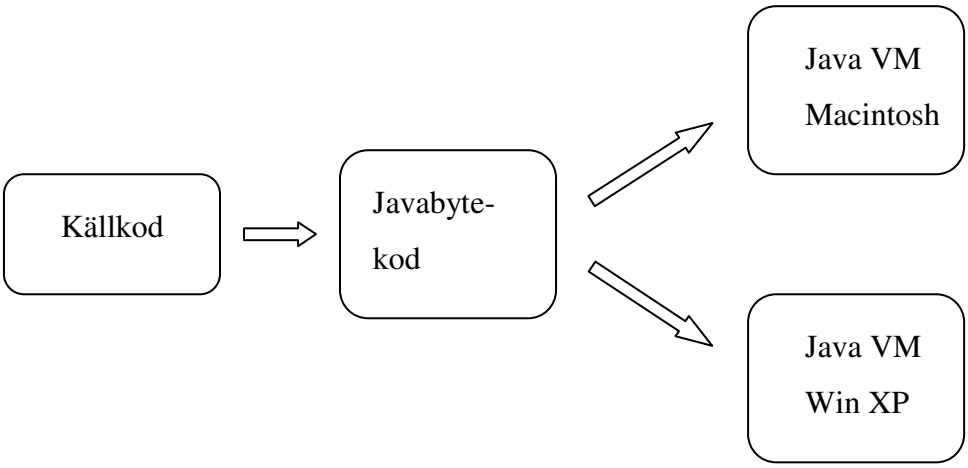
3.1 Java

Java är ett plattformsoberoende programspråk som har utvecklats av Sun Microsystems. Java kallades ursprungligen Oak och utvecklades för att skriva program för konsumentelektronik. Det var då viktigt att språket var plattformsoberoende eftersom olika konsumentelektroniksystem använder sig av många olika typer av processorer. Java lanserades officiellt 1995 och spred sig snabbt mycket tack vare att Java-applets kunde exekveras direkt i en webbläsare.[8]

I följande avsnitt ges en kort introduktion till Java.

3.1.1 Plattformsoberoende

Att Java är plattformsoberoende innebär att man kan exekvera ett javaprogram på olika typer av datorsystem utan att man behöver ändra i programmet. Detta är möjligt eftersom ett javaprogram exekveras i en så kallad virtuell javamaskin. Vid traditionell kompilering översätter kompilatorn källkoden till objektkod. Denna objektkod består av en eller flera objektfiler som sätts samman till en exekverbar fil av länkaren. När programmet sen ska startas placeras den exekverbara filen i primärminnet och programmet exekveras. Som visas i Figur 3-1 skapas javabytekod istället för objektkod vid kompilering av källkod som är skriven i Java. Javabytekoden laddas av den virtuella javamaskinen som ser till att de olika instruktionerna utförs.[7]



Figur 3-1: Olika operativsystem kan exekvera samma källkod

3.1.2 Grafiska verktyg

I Java finns det verktyg för att skapa grafiska användargränssnitt. Detta innebär att det inte krävs extra tillägg för att skriva grafiska program i Java, till skillnad från andra programspråk. Skriver man ett grafiskt program i Windows XP miljö kan man enkelt flytta programmet till en annan plattform utan att de grafiska hjälpfunktionerna man använt sig av måste skrivas om. För att skriva grafiska program i Java till olika system räcker det alltså att lära sig Javas grafiska verktyg.[7]

3.1.3 Andra fördelar

Några andra fördelar med Java är [8]:

- Skräpsamling (Garbage collection)
- Objektorienterat
- SQL-databasstöd

3.1.4 Prestanda

När man pratar om nackdelar med Java pratar man ofta om prestandan. Detta beror till stor del på att Javaprogram interpreteras vid exekvering vilket stjäl en del prestanda i programmet. En annan del i Java som stjäl prestanda är ”skräpsamlingen”. I övrigt har inte Java någon konstruktion som ska påverka prestandan mer negativt än i något annat programspråk.[7]

3.2 SQLite

SQLite är en liten databasmotor som inte kräver en separat serverprocess likt andra SQL-databaser, man säger att SQLite är serverlös. Detta innebär att SQLite kräver minimal konfigurering eftersom man slipper installera, konfigurera och administrera en serverprocess. En SQLite-databas består av en databasfil lagrad direkt på disk till skillnad mot andra SQL-databaser som ofta består av flera filer som endast serverprocessen har rättigheter till. Om en användare har läs-/skrivrättigheter till en SQLite-databasfil kan den läsa resp. skriva i databasen.[9]

SQLite följer SQL92-standarden och har därför stöd för flesta av SQL-kommandon. Det finns dock vissa egenskaper som SQLite inte har stöd för, t ex främmande nycklar. SQLite har också några egenskaper som inte följer standard. Ett exempel på detta är en sorts typlöshet, detta behandlas vidare i avsnitt 3.2.2. SQLite har stöd för JDBC som beskrivs i avsnitt 3.3.[9]

3.2.1 När är det lämpligt att använda SQLite?

SQLite för med sig många fördelar men också nackdelar. Vissa användningsområden är mer lämpliga än andra. Enligt utvecklarna av SQLite fungerar databasmotorn bra till följande användningsområden (enligt [9]):

- Normalbelastade webbsidor
- Små applikationer till t.ex. mobiltelefoner
- Databaspedagogik
- Filformat till applikation
- Övriga applikationer där inte många användare skriver samtidigt till databasen

Det finns också vissa användningsområden då en annan databasmotor är mer lämplig. Dessa är (enligt [9]):

- Klient/Server applikationer
- Högt belastade webbsidor
- Stora datasamlingar
- Övriga applikationer med hög transaktionskonkurrens

3.2.2 Datatyper

De flesta SQL-databaserna använder statisk typning, detta innebär att varje kolumn i en tabell bara kan lagra data av en viss typ. SQLite inför något som de själva kallar för manifest typing. Detta är en sorts typlöshet som innebär att man till en kolumn i en tabell i databasen kan lagra data oavsett vilken datatyp man angett till den aktuella kolumnen vid skapandet av tabellen. Detta innebär i sin tur att man egentligen inte behöver ange några datatyper när man skapar en tabell i en SQLite-databas. Det finns dock ett undantagsfall, när primärnyckeln i tabellen är av typen integer kräver SQLite att den data som man försöker lagra i den kolumnen är av samma typ.[9]

Många anser att manifest typing är en bugg i SQL-sammanhang eftersom språket SQL normalt har statisk typning. Utvecklarna till SQLite ser dock detta som en egenskap.[9]

Alla värden som lagras i en SQLite-databas tilldelas en lagringsklass. De olika lagringsklasserna som finns är [9]:

- NULL, värdet är null.
- INTEGER, värdet är ett heltal.
- REAL, värdet är ett flyttal.
- TEXT, värdet är en textsträng som lagras med databaskodning av typen (UTF-8, UTF-16BE eller UTF-16-LE).
- BLOB, värdet är en mängd data som lagras utan kodning.

För att öka kompabiliteten mellan SQLite och andra databasmotorer har SQLite infört något som de kallar för type affinity. Detta innebär att när man skapar en tabell i en databas kan man ange en rekommenderad datatyp för varje kolumn. Detta är alltså inget krav men det gör det möjligt att spara plats i databasen samt gör det lättare för programmeraren att veta vilken datatyp som är tänkt för kolumnen.[9]

De olika datatyperna som man kan rekommendera när man skapar en tabell är:

- **TEXT**, kan lagra alla data av lagringsklasserna NULL, TEXT och BLOB. Om man försöker lagra numerisk data i en TEXT-kolumn konverterar SQLite den till typen TEXT innan den lagras.
- **NUMERIC**, kan lagra alla typer av lagringsklasser. Vid försök att lagra data av typen TEXT konverteras texten till INTEGER eller REAL.

- **INTEGER**, fungerar på samma sätt som NUMERIC förutom att SQLite inte lagrar decimalerna.
- **NONE,** föredrar ingen lagringsklass före någon annan.

Alla lyckade konverteringar innebär att det sparas plats i databasen.[9]

3.2.3 Minnesallokering

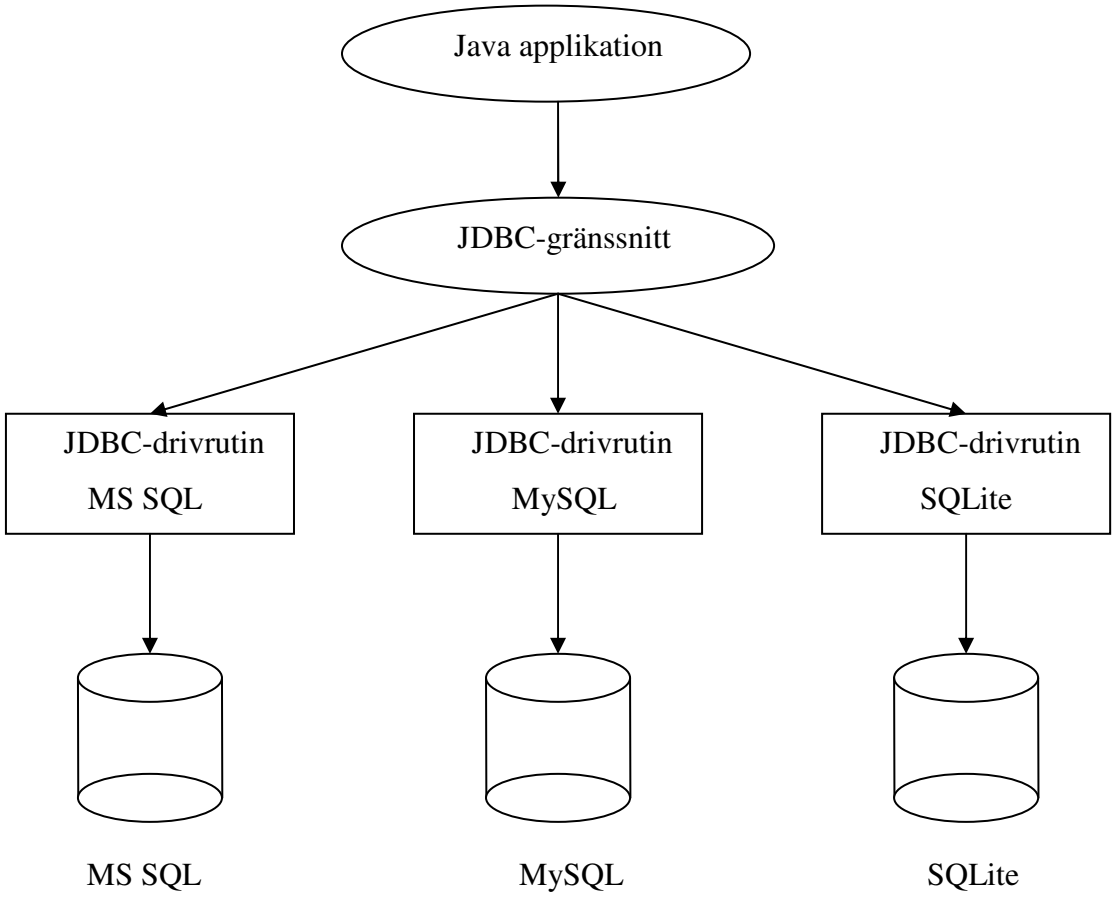
De flesta databasmotorerna allokerar en statisk mängd minne för varje rad i en tabell. Hur mycket minne som allokeras bestäms av hur kolumnerna har deklarerats när man skapat en tabell. Om man t ex deklarerar en kolumn till VARCHAR(40) innebär det att databasmotorn kommer allokera 40 bytes minne till varje textsträng man försöker lagra i den kolumnen, oavsett hur mycket data varje rad innehåller. Om det skulle visa sig att textsträngen man försöker lagra är längre än 40 bytes klipper t ex MySQL textsträngen efter 40 bytes. När man skapar en tabell med samma egenskaper i SQLite behöver man inte ange storleken på kolumnen, vilket innebär att storleken på den data man försöker lagra aldrig tar mer plats än vad som behövs. Det finns dessutom inga begränsningar på hur lång den inmatade textsträngen får vara.[9]

Om man ska lagra data i en kolumn i en SQLite-databas försöker databasmotorn först använda den rekommendera datatypen som angavs vid skapandet av tabellen för den kolumnen. I de fall som datatypen på den data man försöker lagra överensstämmer med den aktuella kolumnens datatyp lagras den i binär form. Detta innebär att lagringsplats sparas i databasen.[9]

3.3 JDBC

JDBC är en förkortning för Java DataBase Connectivity och används för att koppla samma ett program som är skrivet i Java med en databas för att exekvera SQL-frågor. JDBC har ett generellt gränssnitt som man kan skriva anrop emot. Detta gränssnitt tillsammans med att Java är plattformsoberoende gör det möjligt att skriva ett program som både kan exekveras i olika operativsystem och arbeta mot olika typer av databaser utan att behöva göra stora ändringar i källkoden. Ett krav är dock att databasmotorn man använder har stöd för JDBC. Varje databasmotor som har stöd för JDBC har en unik JDBC-drivrutin som används för att översätta anrop från det generella gränssnittet i JDBC till anrop mot den aktuella databasmotorn.

Som beskrivits i avsnitt 3.2 har SQLite stöd för JDBC. Bilaga A visar bland annat exempel på hur anrop mot en databas i SQLite kan göras med hjälp av JDBC.[8]



Figur 3-2: Kopplingen mellan ett javaprogram och en databas via JDBC.

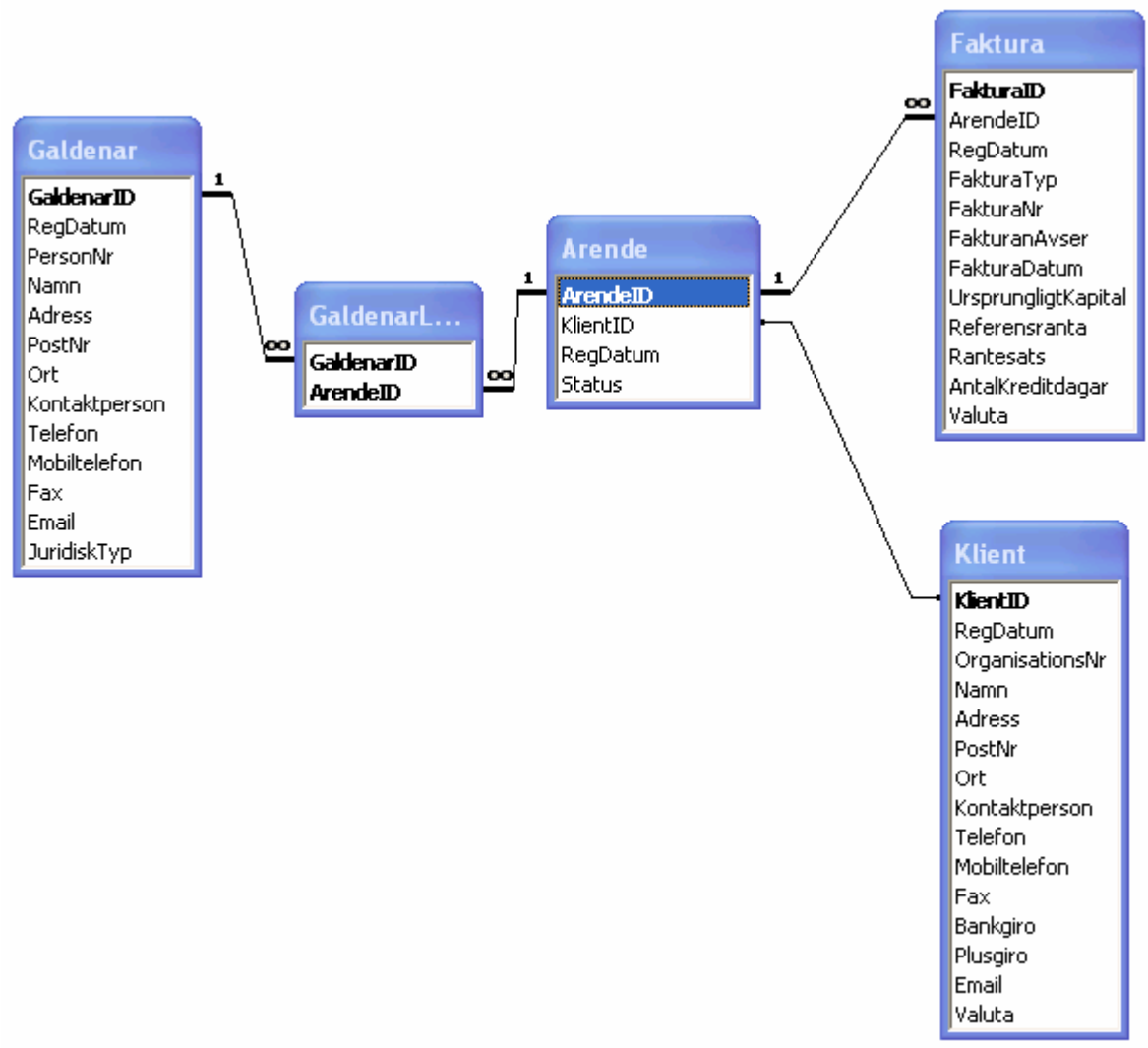
4 Databasdesign

Som beskrivits i kapitel 3 har en SQLite-databas använts för att lagra information i programmet. I det här kapitlet presenteras hur databasen är uppbyggd samt hur den skapades. Avsnitt 4.1 ger en översikt över vilka tabeller som databasen är uppbyggd samt hur databasen skapades. Avsnitt 4.2 beskriver de olika databastabellerna mer ingående.

4.1 Översikt över databasen

Databasen består av fem tabeller där grundtabellen som sammankopplar de övriga är Ärende. De övriga fyra tabellerna är Klient, Gäldenär, GäldenärLista och Faktura. För mer information om vad de olika tabellerna innehåller se Figur 4-1.

För att skapa databasen skrevs ett javaprogram som skapade en tom databas med de olika tabellerna. Koden till detta program visas i bilaga A. Vid skapandet av tabellerna i databasen angavs endast fältnamn, datatyp och primärnyckel. Detta eftersom SQLite allokerar minne dynamiskt och att storleken på de olika fälten då inte behöver anges. SQLite har tyvärr bara stöd för engelskt alfabet vilket innebär att å, ä, ö inte kan användas till tabellnamn och fältnamn. På grund av detta heter tabellen Ärende egentligen Arende. För bättre förståelse har dock å, ä, ö använts vid beskrivning i uppsatsen.



Figur 4-1: Relationsdiagram

4.2 Beskrivning av relationerna/tabellerna

Tabellerna Ärende, Klient, Gäldenär och Faktura har var och en ett fält som unikt identifierar varje rad (tuple) i tabellen. Dessa kallas för tabellens id och är tabellernas primärnyckel. De har logiska namn som ÄrendeID, KlientID och så vidare. Förutom id finns det i dessa tabeller ett fält som heter RegDatum. Detta fält innehåller det datum då ärendet, klienten, gäldenären eller fakturan registrerades i programmet. Datumet är i formatet YYYY-MM-DD.

I avsnitt 4.2.1 till och med 4.2.5 beskrivs de olika tabellerna lite mer ingående. Där kommer inte fälten id och RegDatum att behandlas vidare. För varje databastabell finns det i bilaga B en tabell innehållande de fält som den aktuella databastabellen har. I tabellen finns fältets alla namn, datatyper samt en kort beskrivning om den information som varje fält innehåller.

4.2.1 Ärende

Ärende är den tabell som kopplar samman de andra tabellerna. Den innehåller bland annat id på den som är klient i ärendet samt vad ärendet har för status. KlientID är det fält som identifierar vem som är klient i ärendet. Fältet Status innehåller den status som ärendet kan anta. För tillfället finns två lägen, aktivt och avslutad. För mer information om Ärende se bilaga B.1.

4.2.2 Klient

Tabellen Klient har en 1:M-relation till Ärende. Detta innebär att ett ärende bara kan ha en klient men att en klient kan vara iblandad i flera ärenden. Klienttabellen innehåller information som är direkt kopplad till klienten, såsom organisationsnummer, namn, adress med mera. Se bilaga B.2 för mer information om vilka fält som tabellen innehåller, vilken datatyp de har med mera.

4.2.3 Gäldenär

Tabellen Gäldenär har en N:M-relation till Ärende. Ett ärende kan ha flera gäldenärer men en gäldenär kan också vara inblandad i flera ärenden. När denna typ av relation är mellan två tabeller behövs en mellantabell. I detta fall heter den GäldenärLista och beskrivs i avsnitt 4.2.4. Gäldenärtabellen innehåller information som är direkt kopplad till gäldenären, t.ex. personnummer, namn, adress med mera. För mer information om tabellen Gäldenär, se bilaga B.3.

4.2.4 GäldenärLista

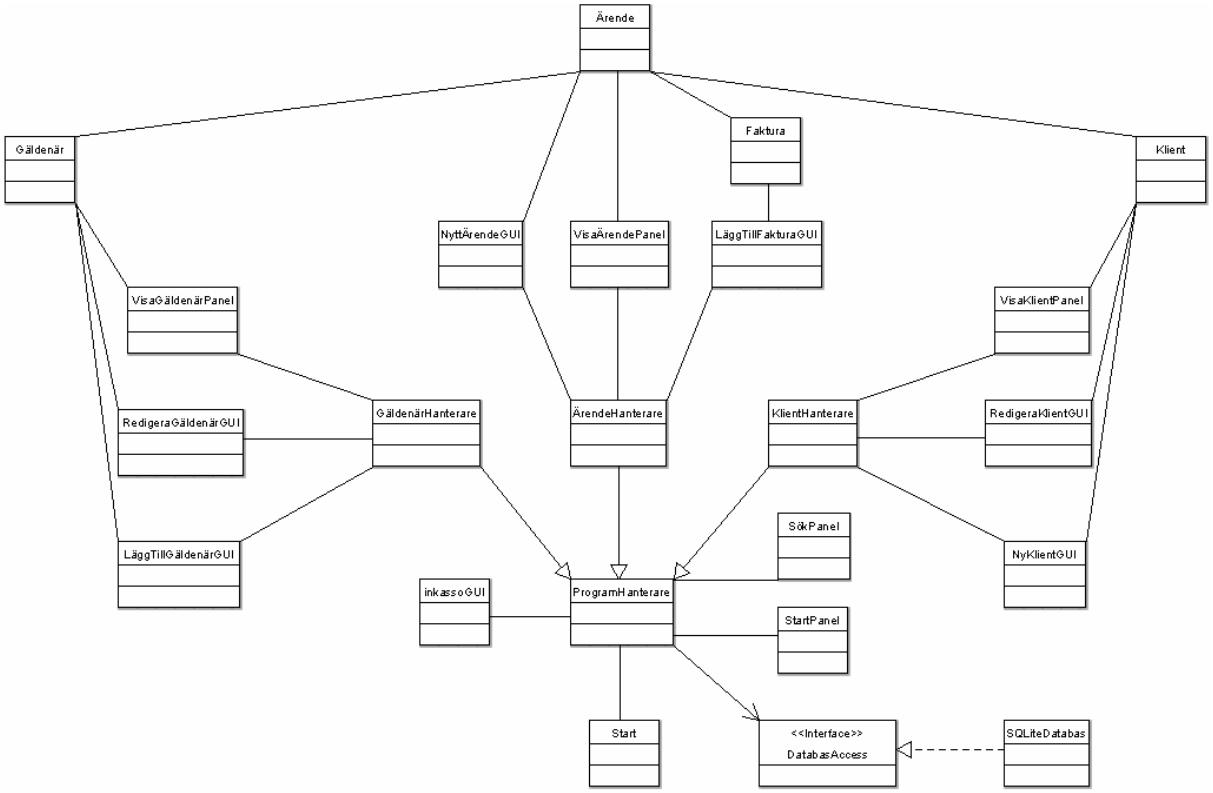
GäldenärLista är den tabell som behövs mellan de båda tabellerna Ärende och Gäldenär eftersom det är en N:M-relation mellan dessa båda tabeller. Denna innehåller primärnycklarna från tabellerna Gäldenär och Ärende. Dessa två nycklar bildar ihop en primärnyckel för tabellen GäldenärLista. För mer information se bilaga B.4.

4.2.5 Faktura

Tabellen Faktura har en 1:M relation till Ärende. Ett ärende kan innehålla flera fakturor men en specifik faktura kan bara kopplas till ett ärende. Fakturatabellen innehåller förutom de som beskrivits i avsnitt 4.2 följande fält: Fakturatyp, FakturaNr, FakturanAvser, FakturaDatum, UrsprungligtKapital, Referensränta, Räntesats, AntalKreditdagar och Valuta. För mer information om vad respektive fält innehåller och av vilken datatyp de är skapta i databasen, se bilaga B.5.

5 Implementationen av programmet

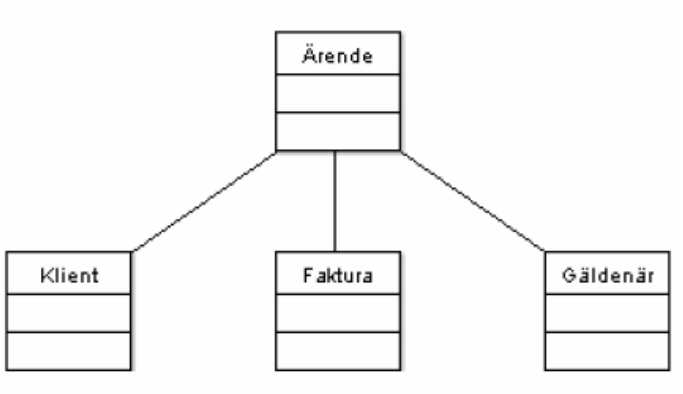
Programmet är uppbyggt i fyra stycken olika delar. Dessa är modell, databas, hanterare och grafiskt användargränssnitt. I det här kapitlet beskrivs de olika delarna samt vilka uppgifter de utför i programmet. Avsnitt 5.1 beskriver vad modellen har för roll i programmet. I avsnitt 5.2 beskrivs hur information lagras ner i databasen. Avsnitt 5.3 behandlar vad hanteraren gör och 5.4 beskriver hur programmets grafiska användargränssnitt är uppbyggt. I Figur 5-1 visas ett klassdiagram över programmet.



Figur 5-1: Klassdiagram över programmet

5.1 Modell

I programmet finns fyra modeller, Ärende, Gäldenär, Klient och Faktura. Modellens uppgift i programmet är att lagra information om varje modells tillstånd i internminnet. För detta innehåller varje modell fält för den information som ska lagras samt metoder för att ändra, hämta och uppdatera informationen. De olika modellerna innehåller bl.a. fält som kan mappas direkt mot fälten i respektive tabell i databasen. För dessa fält finns det metoder för att ändra respektive hämta information i fälten. Eftersom dessa fält redan beskrivits i kapitel 4 kommer endast de övriga fälten och metoderna att behandlas vidare i detta kapitel. I avsnitt 5.1.1 till och med 5.1.3 beskrivs dessa olika modeller mer ingående. Figur 5-2 visar relationerna mellan de olika modellerna.



Figur 5-2: Klassdiagram över modellerna

5.1.1 Ärende

Ärende är den modell där all information om ärendet lagras. Förutom de fält som redan beskrivits i kapitel 4 finns det fält i ärendemodellen som innehåller den klient samt de gäldenärer och fakturor som finns lagrad i ärendet. Eftersom det i ett ärende bara finns en klient lagras den i ett fält av datatypen Klient. Gäldenärerna och fakturorna lagras i fälten gäldenärer respektive fakturor som är av datatypen ArrayList. Dessa två listor innehåller objekt av datatypen Gäldenär respektive Faktura. För att ändra i fälten klient, gäldenärer och fakturor finns det i ärendemodellen metoder för att lägga till eller ta bort en klient, gäldenär eller faktura. Modeller som är lagrade i ärendet kan hämtas genom metoderna hämtaKlient, hämtaGäldenär och så vidare. I ärendemodell finns det fält som innehåller information om ärendets ursprungliga kapital, ursprungliga ränta med mera. Ärendets ursprungliga kapital och ursprungliga ränta är summan av samtliga fakturor i ärendets ursprungliga kapital respektive ursprungliga ränta. Dessa fält uppdateras när man lägger till eller tar bort en faktura.

När det skapas en ny instans av Ärende ändras ärendets status till aktivt och registreringsdatumet till dagens datum, för övrigt nollställs alla fält. Vid varje förändring av modellen kontrolleras ärendets giltighet. För att vara ett giltigt ärende krävs att det innehåller en klient, minst en gäldenär och minst en faktura.

Se bilaga D.1 för klassdiagram över Ärende.

5.1.2 Klient / Gäldenär

Modellerna Klient & Gäldenär innehåller nästan bara fält som är direkt kopplade till klienten respektive gäldenären. Det enda fält som finns utöver dessa är fältet OK som innehåller

information om klienten eller gäldenären är giltig eller inte. I programmet anses en klient eller en gäldenär vara giltig då fälten orgNr/personNr, namn, adress, postNr och ort inte är tomma.

När det skapas en ny instans av Klient/Gäldenär sätts registreringsdatumet till dagens datum, för övrigt nollställs alla fält.

Se bilaga D.2 för klassdiagram över Klient och bilaga D.3 för klassdiagram över Gäldenär.

5.1.3 Faktura

I programmet finns det tre typer av fakturor, debetfaktura, kreditfaktura och räntefaktura. Den vanligaste typen av faktura kallas debetfaktura. Det är sådan som i vardagligt tal brukar kallas för faktura och som innehåller information om att gäldenären ska betala pengar till klienten. Denna faktura kan innehålla dröjsmålsränta enligt vad som beskrivits i 2.1.5. En kreditfaktura är en faktura där klienten betalar tillbaka pengar till gäldenären. I dessa fall blir det ursprungliga kapitalet negativt. Med detta medföljer en negativ ränta som klienten blir skyldig gäldenären. Den tredje typen av faktura som kallas räntefaktura innehåller aldrig någon ränta. I detta fall är alltid räntan 0.

För att lagra ner information om en faktura används modellen Faktura. I avsnitt 4.2.5 beskrevs de fält som databastabellen Faktura innehåller. Förutom dessa innehåller modellen fälten resterandeKapital, ursprungligRänta, resterandeRänta och OK. Fakturans resterande kapital är det ursprungliga kapitalet minus det kapital på fakturan som blivit inbetalat. Eftersom projektet inte har behandlat inbetalningar kommer det resterande kapitalet alltid att vara detsamma som det ursprungliga kapitalet. Samma sak gäller den resterande räntan.

Fältet ursprunglig ränta innehåller den eventuella dröjsmålsränta som tillkommit fakturan. För att beräkna räntan används metoden beräknaRänta som utför beräkning på samma sätt som i exemplet i avsnitt 2.1.5. Vid beräkningen anropas metoden beräknaAntalRäntedagar som beräknar det antalet dagar som fakturans dröjsmålsränta ska beräknas på. Eftersom ränteberäkningen sker i modellen behöver inte någon information om räntan uppdateras i databasen. Likt de övriga modellerna innehåller fältet OK information om fakturan är giltig eller inte. I programmet anses en faktura vara giltig om fälten fakturaNr, fakturanAvser och fakturaDatum inte är tomma. Viktigt att poängtera är att det inte sker någon kontroll av det inmatade fakturadatumet i modellen. Det är alltså användargränssnittets uppgift att se till att information som registreras är av rätt typ. När det skapas en ny instans av faktura ändras fakturatypen till debetfaktura, det ursprungliga kapitalet och räntesatsen till 0, valuta till SEK och referensränta till 1,5. De övriga fälten nollställs. Se bilaga D.4 för klassdiagram över Faktura.

5.2 Databas

Som tidigare beskrivits i kapitel 3 används en databas för att spara ner information i programmet. Databasen som används är en SQLite-databas och är uppbyggd på det sätt som beskrivits i kapitel 4. För att enkelt kunna byta mellan olika databaser skrevs ett databasinterface som databasklassen i programmet implementerar. När en metod i databasen anropas sker det via databasinterfacet. Detta gör att ändringarna i programkoden minskas vid byte av databashanterare. För att se vilka metoder som databasinterfacet innehåller, se bilaga C. Alla anropen mot databashanteraren sker via det generella gränssnittet i JDBC. Detta är också ett sätt som skulle förenkla ett eventuellt byte av databashanterare.

Databasklassen består till största del av fem olika sorters metoder, spara, hämta, uppdatera, ta bort och sök. Dessa kommer att behandlas vidare i avsnitt 5.2.1 till och med 5.2.5. Förutom dessa finns det metoder för att skapa och stänga ner uppkopplingen mot databasen. Ett exempel på hur detta sker i SQLite visas i bilaga A.

5.2.1 Spara

Det finns fyra stycken olika tillfällen när programmet sparar ner information i databasen. Dessa är när ett nytt ärende eller klient registreras samt när man lägger till en ny gäldenär eller faktura. Som beskrivits tidigare i kapitel 4 består databasen av fem stycken olika tabeller, Ärende, Klient, Gäldenär, GäldenärLista och Faktura. I databasklassen finns det metoder för att spara ner information i respektive tabell. Dessa heter sparaÄrendeTabell, sparaKlientTabell etc. och är privata vilket innebär att de inte är synliga utanför databasklassen. Anropet till dessa sker från metoderna sparaÄrende, sparaKlient, läggTillGäldenär och läggTillFaktura. När ny information ska sparas ned i en tabell anges inte något värde till det fält som är primärnyckel. Det behövs inte eftersom primärnyckeln automatiskt får ett värde vid lagring i en SQLite-databas.

Metoden sparaÄrende sparar ner ett helt ärende till databasen. Som beskrivits i avsnitt 5.1.1 består ett ärende av olika delar, klient, gäldenär, faktura samt lite information kring ärendet såsom status med mera. I metoden sparaÄrende skapas först en uppkoppling mot databasen. Efter detta plockas de olika delarna ut ur ärendet och sparas ner i databasen med hjälp av olika metoder. Klienten lagras i metoden sparaKlientTabell, gäldenären i sparaGäldenärTabell och så vidare. Lagringen av ärendet i databasen ses som en stor transaktion. Detta för att undvika att bara halva ärendet sparas när det uppstår ett fel vid lagringen. Om det uppstår ett fel raderas den del av ärendet som lagrats från databasen med

hjälp av att en rollback skickas till databashanteraren. Sker lagringen av ärendet problemfritt skickas en commit.

När ett ärende sparas ner i databasen sker följande steg:

1. Skapar en uppkoppling mot databasen.
2. Startar en transaktion.
3. Sparar ner ärendets klient i databasen.
4. Sparar ner övrig ärendeinformation i databasen.
5. Sparar ner ärendets gäldenärer i databasen..
6. Sparar ner ärendets fakturor i databasen.
7. Avslutar transaktionen.
8. Stänger ner uppkopplingen mot databasen.

Metoderna sparaKlient, läggtillGäldenär och läggTillFaktura fungerar nästan på samma sätt som sparaÄrende. Skillnaden är den att sparaÄrende sparar ner ett helt ärende med klient, gäldenär och faktura i databasen medan de övriga lagringsmetoderna endast sparar en specifik klient, gäldenär eller faktura.

I de metoder som utför lagringen skapas en sql-fråga som innehåller den information om ärendet, klienten, gäldenären eller fakturan som ska sparas ned i databasen. Efter detta exekveras sql-frågan. Om det skulle uppstå något fel vid lagringen returnerar metoden false, annars true.

5.2.2 Sök

För att komma åt den information som sparats ner i databasen finns det tre olika sökmetoder i databasklassen, dessa är sökÄrende, sökKlient och sökGäldenär. Som inargument har alla tre ett sökindex och ett sökvärde. Sökindex är den egenskap hos ärendet, klienten eller gäldenären som användaren har valt att söka på. Datatypen på returvärdet från metoderna är ArrayList. Eftersom de tre metoderna fungerar på samma sätt beskrivs endast metoden sökGäldenär. Den arbetar i följande steg:

1. Skapar en uppkoppling mot databasen.
2. Skapar en sql-fråga innehållande de sökkriterier som metoden fick som inargument.
3. Exekverar sql-frågan och tar emot id på de gäldenärer som överensstämde med sökkriterierna.

4. Hämtar alla gäldenärer med de id som returnerades vid sökningen. För detta används hämtaGäldenär som beskrivs i avsnitt 5.2.3.
5. Stänger ner uppkopplingen mot databasen.
6. Sparar ner alla gäldenärer i en ArrayList och returnera den.

I de fall då inte sökningen ger några träffar returnerar metoden null.

5.2.3 Hämta

När information ska hämtas från databasen till programmet används metoderna hämtaÄrende, hämtaKlient och så vidare. Exempel på när detta sker i programmet är vid sökning. För att t.ex. hämta en klient används det id som unikt identifierar varje klient som inargument. I metoden skapas det då en sql-fråga som hämtar all information om den aktuella klienten från databasen och lagrar det i en klientmodell. Om det skulle uppstå ett fel och sql-frågan inte skulle returnera något resultat, returnerar metoden null. I annat fall returneras den modell som skapats. Detta är gemensamt med alla metoder som hämtar information från databasen. Metoderna hämtaGäldenär och hämtaFaktura fungerar på samma sätt som hämtaKlient fast med skillnaden att de returnerar en gäldenärmodell respektive fakturamodell istället för en klientmodell. För att hämta ett ärende finns metoden hämtaÄrende. När denna metod anropas sker följande steg:

1. Skapar en uppkoppling mot databasen.
2. Hämtar med hjälp av ärendeid all information från ärendetabellen i databasen och lagrar den i en ärendemodell.
3. Hämtar med hjälp av det klientid som kommer från steg 2 den klient som finns i ärendet och lagrar ner den i ärendemodellen.
4. Hämtar med hjälp av ärendeid alla gäldenärer som är kopplade till ärendet och lagrar ner dessa i ärendemodellen.
5. Hämtar med hjälp av ärendeid alla fakturor som är kopplade till ärendet och lagrar ner dessa i ärendemodellen.
6. Stänger ner uppkopplingen mot databasen.
7. Returnerar ärendet.

5.2.4 Uppdatera

I programmet finns det möjlighet att redigera befintliga gäldenärer och klienter. Vid dessa tillfällen behöver information i databasen uppdateras. I databasklassen i programmet finns det metoder för detta som heter uppdateraGäldenär, uppdateraKlient och uppdateraÄrende. Eftersom dessa tre metoder fungerar på samma sätt visas endast hur en av metoderna fungerar. När information om en befintlig gäldenär ska uppdateras anropas metoden uppdateraGäldenär. Som inargument till metoden skickas den ändrade gäldenärsmodellen. Uppdateringen sker i följande steg:

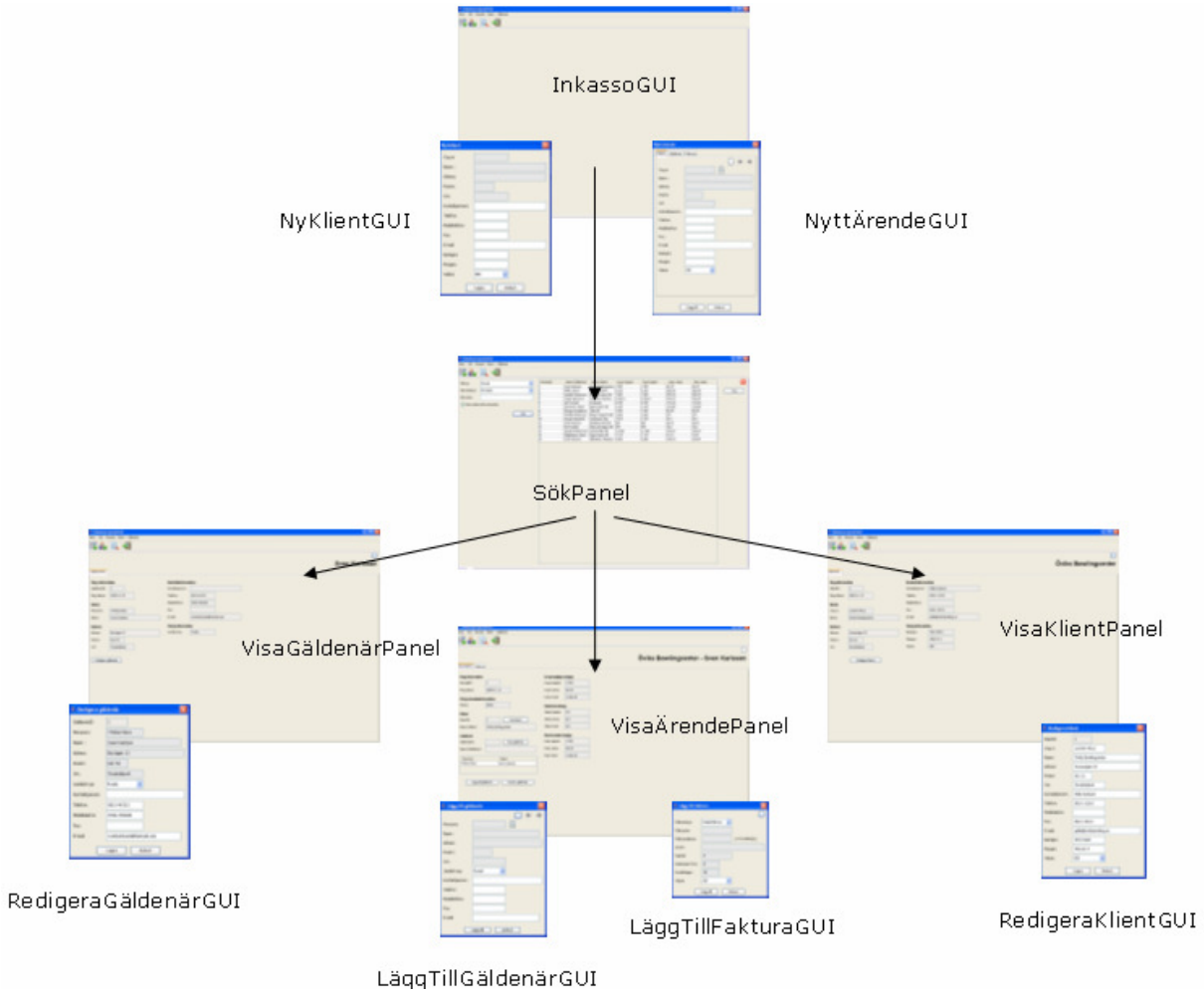
1. Skapar en uppkoppling mot databasen.
2. Startar en transaktion
3. Skapar en sql-fråga som uppdaterar information i databasen kring den gäldenär som ska uppdateras.
4. Exekverar sql-frågan och kontrollerar svaret
5. Avslutar transaktionen
6. Stänger ner uppkopplingen mot databasen.
7. Om resultatet av frågan blir 0 har inte ändringen blivit utförd i databasen och false returneras. I annat fall returneras true.

5.2.5 Ta bort

Vid två tillfällen i programmet behöver information plockas bort ur databasen. Dessa är när användaren väljer att ta bort en gäldenär eller en faktura ifrån ett ärende. Metoderna som sköter detta i databasklassen heter taBortGäldenär respektive taBortFaktura. Skillnaden på dessa två metoder är den att när en gäldenär raderas från ett ärende försvinner bara kopplingen mellan ärendet och gäldenären i databasen. När en faktura raderas försvinner den fysiskt från databasen. Det är alltså möjligt att söka fram gäldenären och koppla den till ett annat ärende. Kopplingen som är mellan ärendet och gäldenären finns i mellanlistan som heter GäldenärLista. Som beskrivits i avsnitt 4.2.4 innehåller den gäldenärens id och ärendets id som på så sätt kopplar dem samman. Metoden taBortGäldenär tar från GäldenärLista bort den tuple innehållande den aktuella gäldenärens id och id på det ärendet som denna ska raderas från. När en faktura ska raderas tar metoden taBortFaktura bort den tuple innehållande fakturans unika id.

5.3 Grafiskt användargränssnitt (GUI)

Programmet grafiska användargränssnitt är uppdelat i tre stycken olika vyer. Dessa är huvudvy, paneler och formulär. Programmets huvudvy heter InkassoGUI och beskrivs i avsnitt 5.3.1. Panelerna beskrivs i avsnitt 5.3.3 och formulären 5.3.4. Panelerna och formulären använder objekt av klassen StandardPanel. Denna klass beskrivs i avsnitt 5.3.2. Hur de olika delarna i det grafiska gränssnittet hänger ihop beskrivs i Figur 5-3.

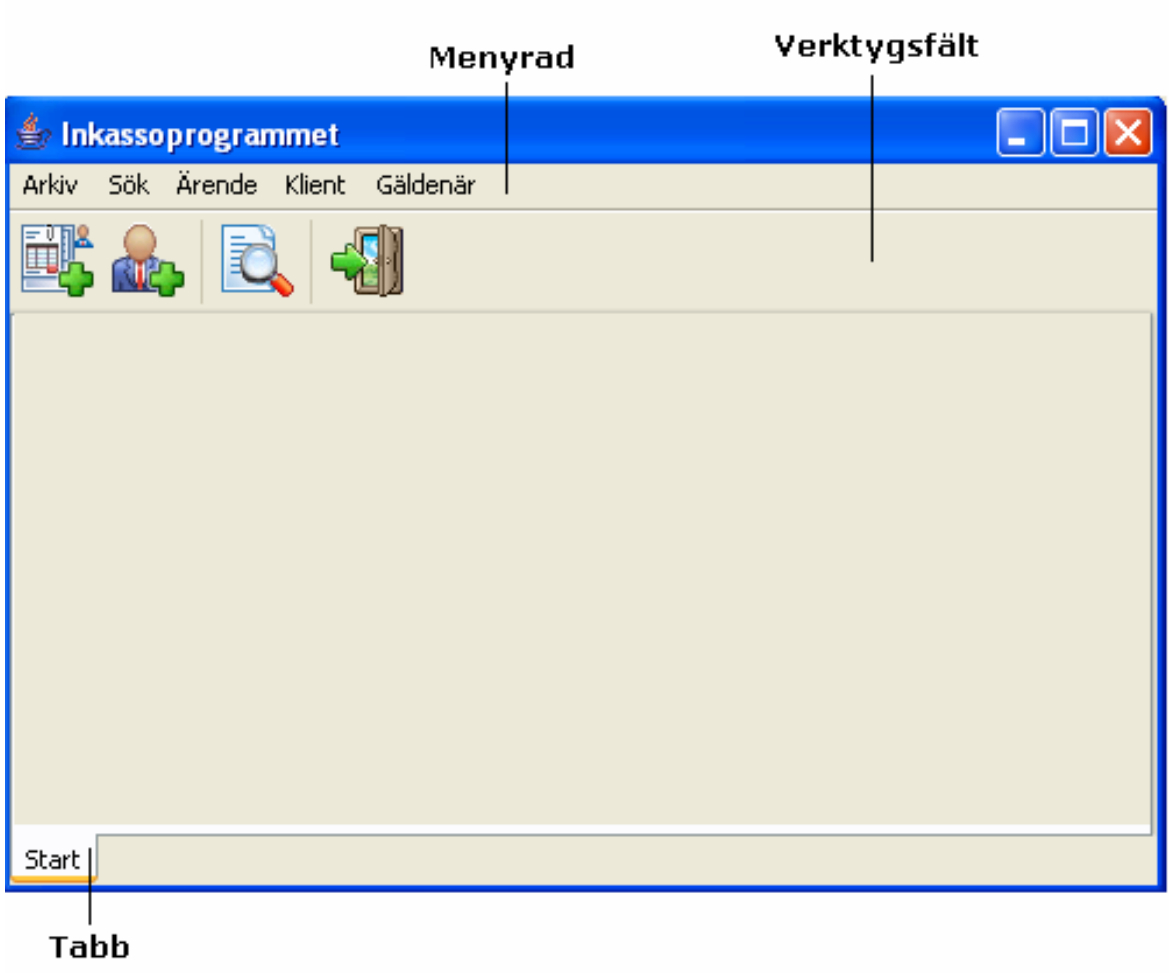


Figur 5-3: Fönsternavigeringsdiagram

5.3.1 InkassoGUI

Ett av målen med projektet var att till programmet skapa ett grafiskt användargränssnitt som både var lätt att överskåda och använda. Som Figur 5-4 visar har GUI:t ett klassiskt windowsutseende med en menyrad längst upp där användaren kan navigera mellan de olika delarna i programmet. För detta finns det också ett verktygsfält med knappar som är kopplade till de vanligaste menyvalen. Längst ner i Figur 5-4 visas att fönstret är uppbyggt av tabbar.

När användaren väljer ett menyval som är kopplat till en panel visas den som en ny tabb i GUI:t. Vilka menyval som går att klicka på styrs av vilken typ av panel som är aktiv i tabbpanelen. De olika paneltyperna beskrivs i avsnitt 5.3.3. Tabbarna gör det enkelt för användaren att navigera när flera paneler är öppna samtidigt. När användaren klickar på ett menyval som är kopplat till ett formulär visas en modal dialogruta med det aktuella formuläret. Det är då inte möjligt att komma åt några andra delar av programmets GUI än just formuläret.



Figur 5-4: Bild på InkassoGUI

5.3.2 StandardPanel

För att skapa en komponent, sätta egenskaper till den och placera den i en kontainer krävs många kodrader. När ett GUI som ofta i detta fall är uppbyggt av många komponenter blir GUI-klasserna väldigt stora. Eftersom de flesta GUI i programmet har samma uppbyggnad skapades klassen StandardPanel. StandardPanel ärver egenskaper från javas standardklass

JPanel som är en kontainer dit komponenter kan placeras. I klassen StandardPanel finns det metoder för att skapa och hämta komponenter. De komponenter som går att skapa är:

- Etikett (JLabel)
- Textfält (JTextField)
- Textruta (JTextArea)
- ComboBox (JComboBox)
- Tabell (JTable)
- Knapp (JButton)
- CheckBox (JCheckBox)

Komponenterna skapas i de metoder som beskrivs i bilaga E. Till varje metod som skapar en komponent skickas en del egenskaper som inargument. När komponenten skapas får den dessa egenskaper innan den placeras på panelen och lagras ner i en HashMap. Varje komponent har ett unikt namn som används när den lagras i HashMapen. Samma namn används när en komponent ska hämtas från standardpanelen. När detta ska göras anropas metoderna hämtaTabell, hämtaKnapp och så vidare. Klassen StandardPanel innehåller också metoder för att aktivera respektive inaktivera de olika komponenterna.

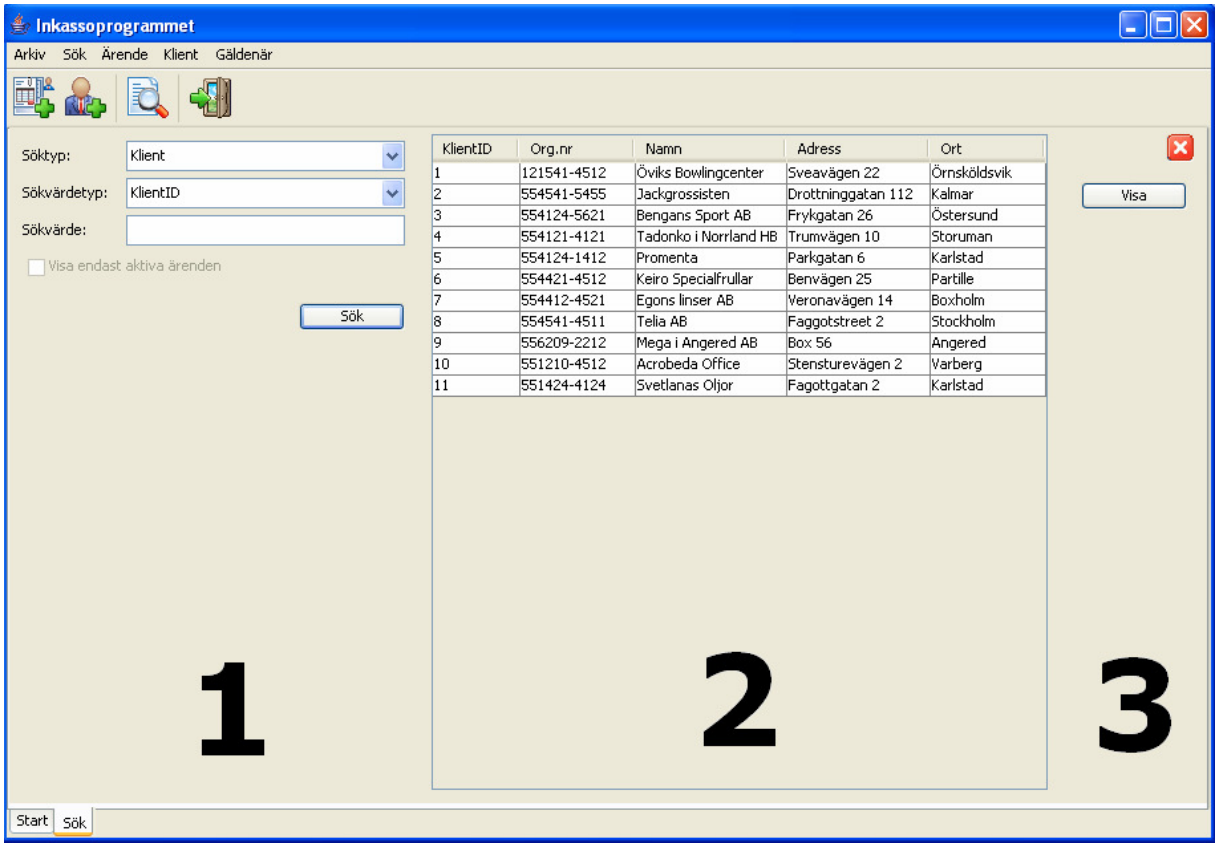
StandardPanel använder sig av en LayoutManager som heter som heter RiverLayout. RiverLayout är en enkel men kraftfull LayoutManager som använder sig av html-inspirerade kommandon för att placera komponenter på t.ex. en panel. Detta gör det relativt lätt att skriva ett enkelt och överskådligt GUI. RiverLayout passar utmärkt för att t.ex. skapa formulär vilket gör att den passar bra i detta program.

5.3.3 Panel

Som beskrivits i avsnitt 5.3.1 är GUI:t bland annat uppbyggt av tabbar. I varje tabb visas någon av programmets olika paneler. De olika panelerna som finns är SökPanel, VisaGäldenärPanel, VisaKlientPanel och VisaÄrendePanel. Det finns också en femte panel som heter StartPanel. Den har som det är nu ingen uppgift i programmet och kommer därför inte att behandlas vidare. Gemensamt med panelerna är att de ärver egenskaper från JPanel och att de har BorderLayout som LayoutManager. BorderLayout använder sig av de fyra olika väderstrecken samt center för att placera komponenter. I de olika delarna av panelen placeras objekt av typen StandardPanel som beskrivits i avsnitt 5.3.2. Figur 5-5 visar uppbyggnaden av en panel, i detta fall SökPanel. Den har tre standardpaneler placerade i väst (1), centralt (2)

och öst (3). En annan sak som är gemensam för panelerna är den knapp som finns i övre högra hörnet. Den används för att stänga en panel. För detta går det också bra att högerklicka på tabben man vill stänga och välja stäng.

De fyra olika panelerna beskrivs mer ingående i avsnitt 5.3.3.1 till och med 5.3.3.4.



Figur 5-5: Uppbyggnaden av sökpanelen

5.3.3.1 SökPanel

När användaren vill söka efter ett ärende, en klient eller en gäldenär används panelen SökPanel. För att komma till sökpanelen klickar man på Sök i menyn eller på knappen med ett förstoringsglas i verktygsfältet. Det går bara att öppna en sökpanel åt gången. Försöker användaren öppna ytterligare en sökpanel visas den panel som redan finns i tabbpanelen. I sökpanelen finns två comboboxar för att välja söktyp och sökvärdetyp. De olika söktyperna är Ärende, Klient eller Gäldenär. Sökvärdetyp är den egenskap hos ärendet, klienten eller gäldenären som man vill söka på. Under comboboxarna finns ett textfält för sökvärde och en checkbox. Checkboxen är bara tillgänglig vid sökning på ärende och anger om sökningen endast ska visa aktiva ärenden eller inte. För att utföra en sökning klickar man på knappen sök. När sökningen är utförd returneras en ArrayList från databasklassen med sökningens träffar. Träffarna visas i den tabell som finns i sökpanelen. För att titta närmare på något av de

sökta ärendena, klienterna eller gäldenärerna kan användaren dubbelklicka på den önskade raden i tabellen. Det går också att välja en rad och klicka på knappen eller menyvalet Visa. Skulle inte någon rad vara vald visas ett felmeddelande. Beroende på om användaren har valt att visa ett ärende, en klient eller en gäldenär visas någon av panelerna VisaÄrende, VisaKlient och VisaGäldenär. SökPanel plockar då ut vad som ska visas från den ArrayList som returnerades vid sökningen och skickar med det till panelen som ska visa. En bild på Sökpanelen visas i Figur 5-5.

5.3.3.2 VisaÄrendePanel

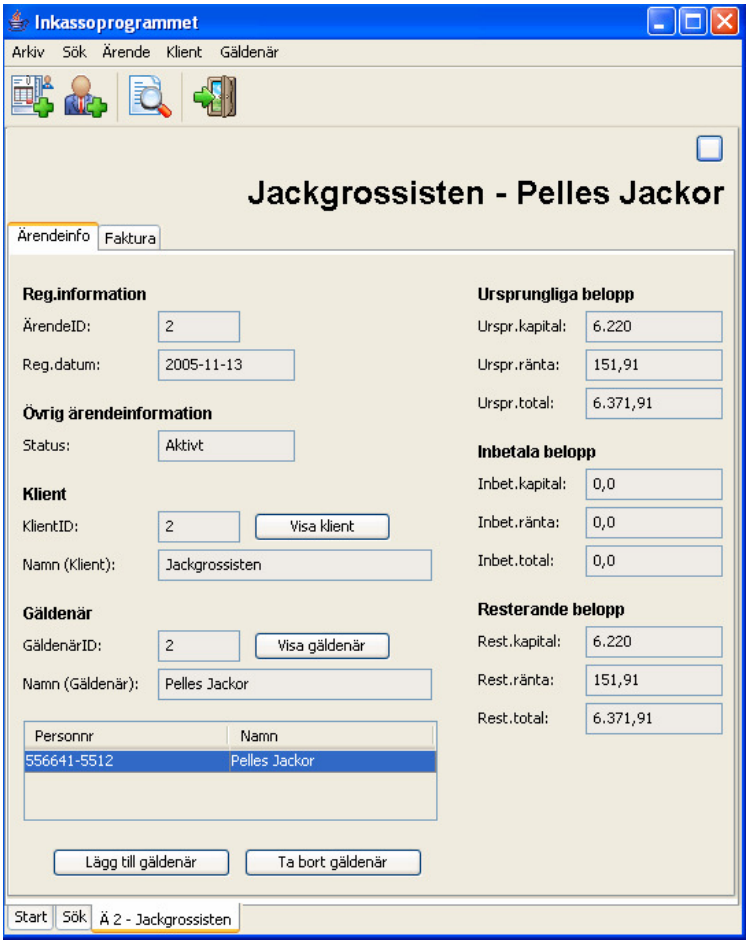
När ett ärende ska visas används panelen VisaÄrendePanel. VisaÄrendePanel är uppbyggd av två paneler, en panel som visar ärendets klient och huvudgäldenär samt en tabbpanel med två tabbar, Ärendeinfo och Faktura. Ärendeinfo är uppdelad i sju stycken delar. I varje del finns en beskrivningsetikett och textfält med information kring den delen av ärendet. I delarna Klient och Gäldenär finns det dessutom en tabell och knapparna Visa klient, Visa gäldenär, Lägg till gäldenär och Ta bort gäldenär. Tabellen visar de gäldenärer som är inblandade i ärendet. Följande händer om man klickar på någon av knapparna:

- **Visa klient** – Plockar ut den klient som hör till ärendet och visar den genom panelen VisaKlientPanel.
- **Visa gäldenär** – Plockar ut den gäldenär som är vald i tabellen och visar den genom panelen VisaGäldenärPanel. Om inte någon gäldenär är vald i tabellen visas ett felmeddelande.
- **Lägg till gäldenär** – Visar formuläret LäggTillGäldenär och uppdaterar VisaÄrendePanelen när formuläret försvinner.
- **Ta bort gäldenär** – Tar bort den gäldenär som är vald i tabellen ifrån ärendet. Om inte någon gäldenär är vald i tabellen eller om ärendet bara innehåller en gäldenär visas ett felmeddelande. Efter att en gäldenär har tagits bort uppdateras VisaÄrendePanelen.

Tabben Faktura är precis som Ärendeinfo uppdelad i mindre delar med beskrivningstext och textfält som innehåller information om en specifik faktura. Förutom det finns det två knappar och en tabell som innehåller ärendets alla fakturor. Knapparna är Lägg till faktura och Ta bort faktura. Följande händer om man klickar på någon av knapparna:

- **Lägg till faktura** – Visar formuläret LäggTillFaktura och uppdaterar VisaÄrendePanelen när formuläret försvinner.
- **Ta bort faktura** – Tar bort den fakturan som är vald i tabellen ifrån ärendet. Om inte någon faktura är vald i tabellen eller om ärendet bara innehåller en faktura visas ett felmeddelande. Efter att en faktura har tagits bort uppdateras VisaÄrendePanelen.

I utgångsläget visar textfälten i fakturatabben information om den första fakturan i ärendet. För att se information om övriga fakturor kan användaren klicka på de andra fakturorna i tabellen. VisaÄrendePanelen visas i Figur 5-6.

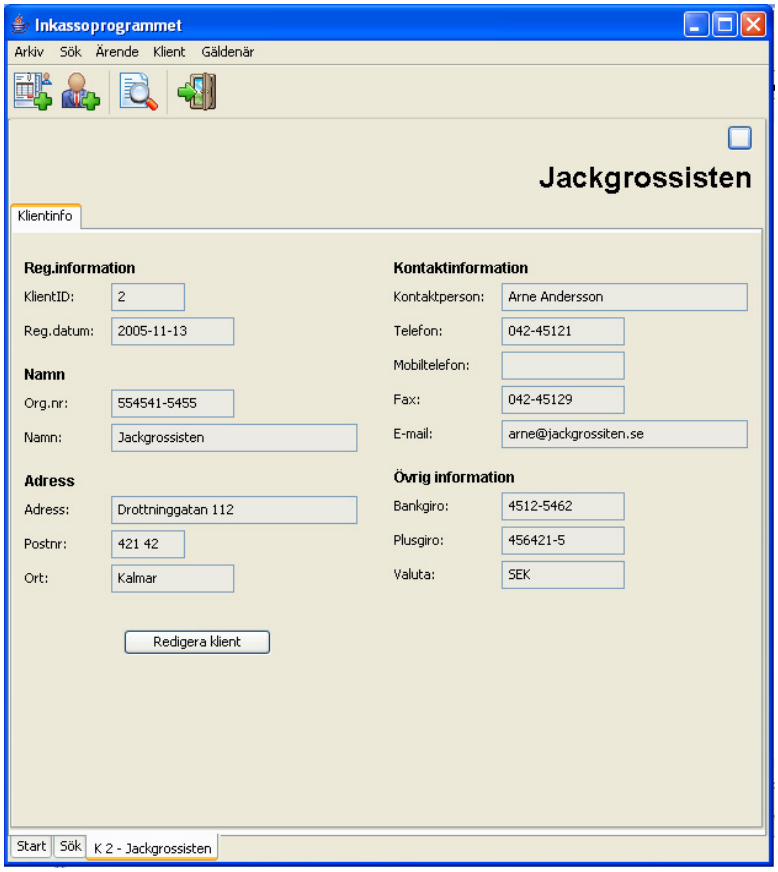


Figur 5-6: Bild på VisaÄrendePanel

5.3.3.3 VisaKlientPanel

Panelen som visar information om en specifik klient heter VisaKlientPanel. Den innehåller textfält som visar information om klienten. Den informationen som visas är den som går att hämta från klientmodellen, t.ex. klientid, namn, adress och så vidare. På panelen finns

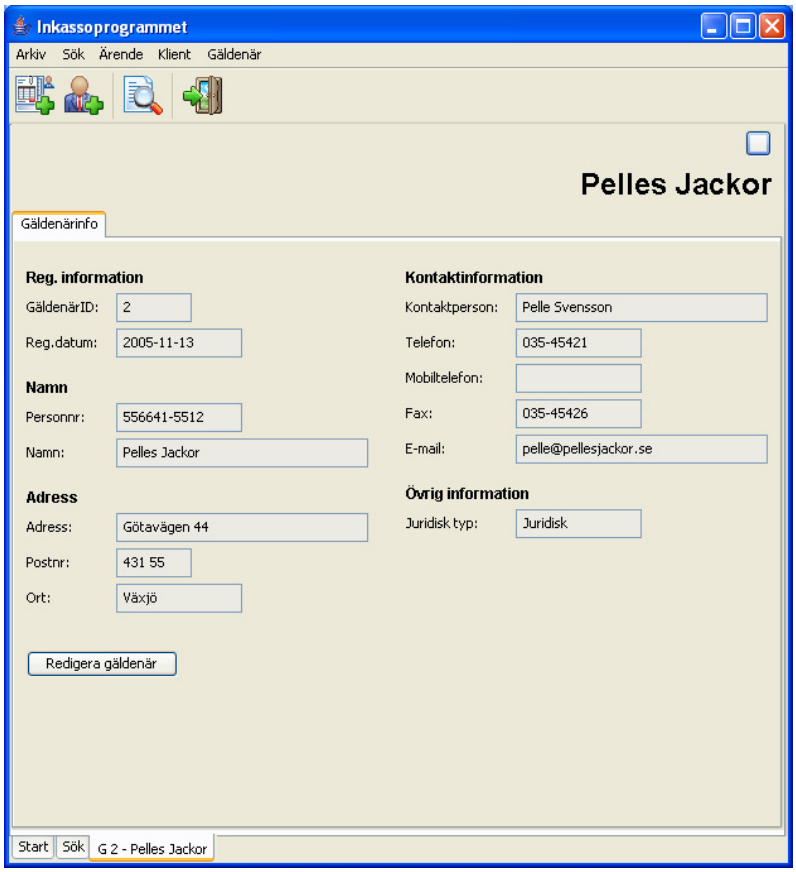
förutom textfälten också knappen Redigera klient. När användaren klickar på knappen skickas den klient som visas till formuläret RedigeraKlientGUI. När formuläret stängs uppdateras VisaKlientPanelen med den nya klientinformationen.



Figur 5-7: Bild på VisaKlientPanel

5.3.3.4 VisaGäldenärPanel

När information om en specifik gäldenär ska visas används panelen VisaGäldenärPanel. Den är uppbyggd på samma sätt som VisaKlientPanelen med textfält för att beskriva gäldenärens egenskaper samt en knapp. Knappen på panelen är Redigera gäldenär. När användaren klickar på knappen skickas gäldenären till formuläret RedigeraGäldenärGUI. När formuläret stängs uppdateras VisaGäldenärPanelen.



Figur 5-8: Bild på VisaGäldenärPanel

5.3.4 Formulär

Det som i programmet kallas för formulär är ett modalt fönster som används för att registrera in information. Formulären ärver egenskaper från javas standardklass JDialog. I programmet finns sex stycken olika formulär. Dessa är NyttÄrendeGUI, NyKlientGUI, LäggTillGäldenärGUI, LäggTillFakturaGUI, RedigeraKlientGUI och RedigeraGäldenärGUI. Gemensamt för formulären är att de är uppbyggda av en StandardPaneler med en mängd textrutor och comboboxar där information kan matas in. Varje formulär har en knapp för att avbryta och stänga ner fönstret samt en knapp för att utföra registreringen/redigeringen. När knappen för att registrera/redigera trycks in på något av formulären kontrolleras först ärendets, klientens, gäldenärens eller fakturans giltighet innan den lagras ned i databasen. Skulle något fel uppstå visas ett felmeddelande.

I programmet är det användargränssnittets ansvar att informationen som matas in av användaren är av korrekt slag. Ett exempel på detta är när fakturadatum på en faktura matas in. I programmet måste datumet ha strukturen ÅÅÅÅ-MM-DD för att ränteberäkningen ska fungera korrekt. Det är då viktigt att GUI:t verkligen ser till att användaren matar in korrekt information. I vissa fall är information obligatorisk. För att användaren ska veta vilka

textrutor som är obligatoriska markeras de med en grå bakgrundsfärg. I avsnitt 5.3.4.1 till och med 5.3.4.6 beskrivs de olika formulären mer ingående. Ett exempel på ett formulär visas i Figur 5-9.

5.3.4.1 NyttÄrendeGUI

När ett ärende ska registreras i programmet används formuläret NyttÄrendeGUI. NyttÄrendeGUI har en ärendemodell där all information kring ärendet lagras. Som Figur 5-9 visar är den uppbyggd av en tabbpanel med tre tabbar. De tre tabbarna är klient, gäldenär och faktura. Varje tabb är uppbyggd av en StandardPanel som var och en innehåller olika komponenter. Klienttabben består av följande komponenter:

- Textrutor och en combobox där information om klienten matas in.
- Hjälptexter till textrutorna.
- Knapp för att tömma alla fält i klienttabben.
- Knapp för att söka på befintliga klienter. Sökvärdet som används är klientens organisationsnummer. Sökfunktionen fungerar på samma sätt som beskrivits i avsnitt 5.3.3.1. Genererar klientsökningen i några träffar visas informationen om den klient som finns först i den ArrayList som databasklassen returnerar.
- Två stycken knappar för att navigera mellan de olika klienterna om en sökning skulle returnera fler än en klient.

Gäldenärtabben är uppbyggd på samma sätt som klienttabben. Eftersom ett ärende kan innehålla fler än en gäldenär finns ytterligare några komponenter i gäldenärtabben. Dessa är:

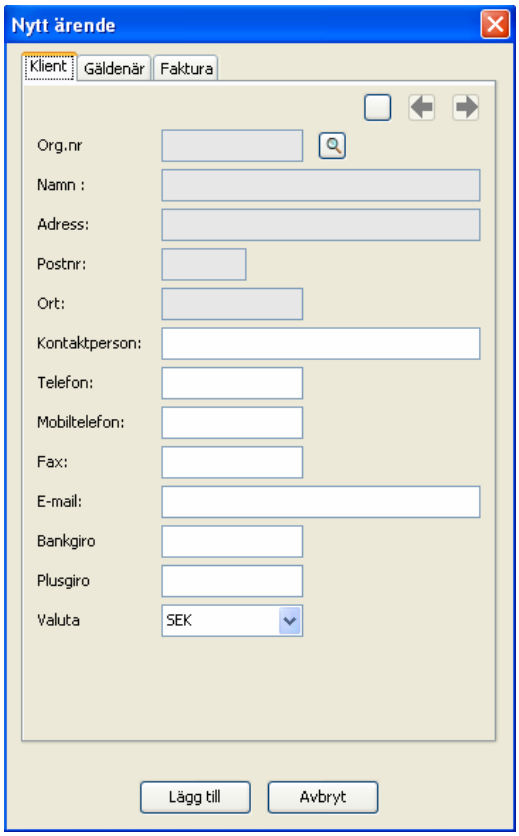
- Tabell för att visa de gäldenärer som är registrerade till ärendet
- Knapp för att lägga till en inmatad gäldenär till ärendet. Klickar användaren på den lagras inmatad information ner i en gäldenärsmodell. Därefter kontrolleras gäldenärens giltighet. För att inte samma gäldenär ska kunna vara registrerad flera gånger i samma ärende kontrolleras detta. Om gäldenären är giltig och inte finns med i ärendemodellen sedan tidigare lagras gäldenären ner i ärendemodellen och placeras i tabellen. I annat fall visas ett felmeddelande.
- Knapp för att ta bort en registrerad gäldenär från ärendet. När användaren klickar på knappen raderas den i tabellen markerade gäldenären från ärendemodellen. Om ingen gäldenär är markerad i tabellen visas ett felmeddelande.

Som beskrivits i avsnitt 4.2.5 kan en faktura bara vara kopplad till ett ärende. På grund av detta finns det inga sökfunktioner i fakturatabben. Ett ärende kan ju däremot innehålla flera fakturor. Av denna anledning finns det knappar för att lägga till och ta bort fakturor från ärendet samt en tabell som visar de fakturor som är registrerade till ärendet. Knapparna lägg till och ta bort fungerar på samma sätt som de knappar som finns i gäldenärtabben. Som beskrivits tidigare i detta avsnitt är det användargränssnittets ansvar att se till information som matas in är av rätt slag. De textrutor som finns i tabbarna klient och gäldenär behöver inga kontroller. Det behövs däremot till några av de textrutorna som finns i fakturatabben. Gemensamt för de rutor som kontrolleras är att kontrollen sker när användaren lämnat en rutan. Dessa textrutor beskrivs mer ingående. Fakturatabben innehåller förutom de som redan beskrivits följande komponenter:

- Knapp för att tömma alla rutor i fakturatabben.
- Combobox för att välja fakturatyp. Som beskrivits i avsnitt Faktura5.1.3 finns det i programmet tre typer av fakturor, debetfaktura, kreditfaktura och räntefaktura. Vid byte av fakturatyp töms alla fält i fakturatabben.
- Textrutor utan kontroll. Dessa är Fakturanr och Avser.
- Textrutan Fakturadatum som kontrollerar att fakturadatumet är uppbyggt på rätt sätt. Datumet ska vara uppbyggt enligt ÅÅÅÅ-MM-DD (t ex 2005-10-21). Om användaren matar in ett fakturadatum av typen ÅÅMMDD (t ex 051021) konverteras det automatiskt till rätt typ. I annat fall töms rutan helt.
- Textrutan Kapital som kontrollerar om användaren matat in ett giltigt belopp. Ett giltigt belopp innehåller bara siffror och eventuellt en punkt för att dela upp heltal och decimaler. Kapitalrutan påverkas av vilken fakturatyp som användaren valt. Vid debetfaktura och räntefaktura accepteras bara positiva belopp. Har användaren däremot valt kreditfaktura som fakturatyp accepteras både positiva och negativa belopp. Detta eftersom ett eventuellt positivt belopp automatiskt ändras till negativt när det kontrolleras. Om ett belopp inte är giltigt ändras innehållet i rutan till sitt ursprungliga värde som är 0.
- Textrutan Räntesats som kontrollerar räntesatsen som matats in är korrekt. Den måste vara ett positivt tal och vara mellan 0 och 100. Precis som vid kapitalbeloppet accepterar decimaltal. Om räntesatsen inte är korrekt ändras den till sitt ursprungliga värde som är 8.

- Textrutan Kreditdagar som kontrollerar antalet kreditdagar är korrekt. Kraven för denna ruta är att den innehåller ett positivt heltal. Om detta inte är fallet ändras innehållet till sitt ursprungliga värde som är 30.
- Comboboxen för att välja vilken valuta som beloppen på fakturan är baserad på.

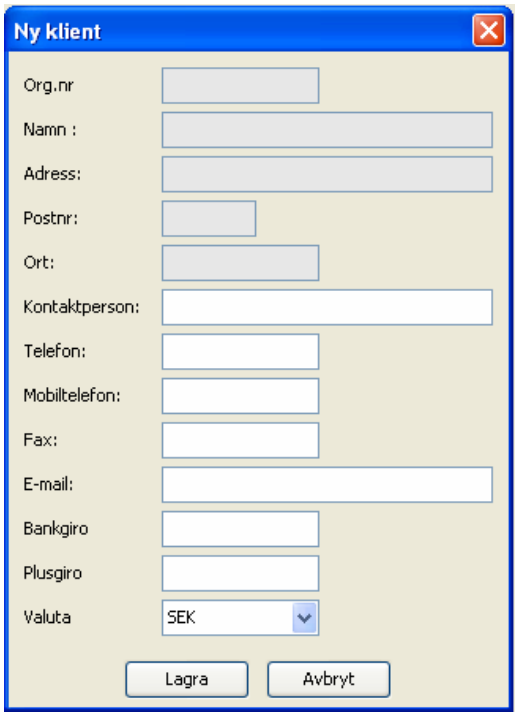
Förutom de tre tabbarna som beskrivit finns i NyttÄrendeGUI två knappar. Dessa är Lägg till och Avbryt. Om användaren trycker på knappen Avbryt stängs formuläret ner. Knappen Lägg till används när all information till ärendet är inregistrerat. Trycker användaren på den kontrolleras ärendemodellen giltighet. Om ärendemodellen är giltig lagras den ner i databasen med hjälp av ärendehanteraren. I annat fall visas ett felmeddelande. Skulle något fel uppstå vid lagringen i databasen visas ett felmeddelande som visar detta. Går däremot allting som det ska stängs formuläret ner.



Figur 5-9: Bild på NyttÄrendeGUI

5.3.4.2 NyKlientGUI

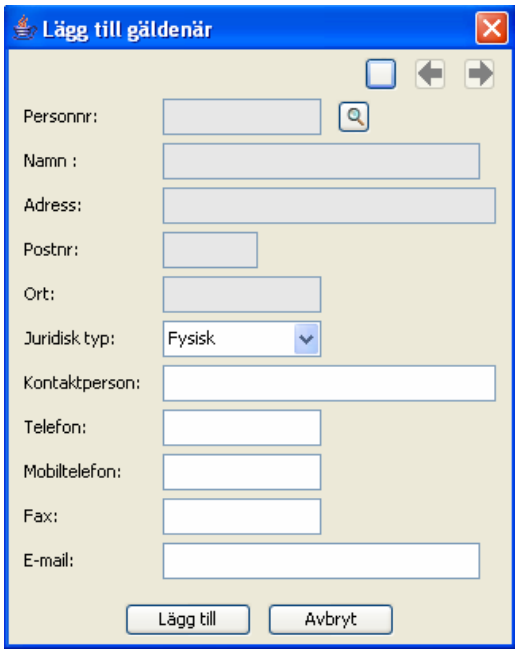
Formuläret NyKlientGUI används när en ny klient ska registreras in i programmet. Det har en mängd textrutor och en combobox där användaren kan mata in information kring klienten som ska registreras. En bild på NyKlientGUI visas i Figur 5-10.



Figur 5-10: Bild på NyKlientGUI

5.3.4.3 LäggTillGäldenärGUI

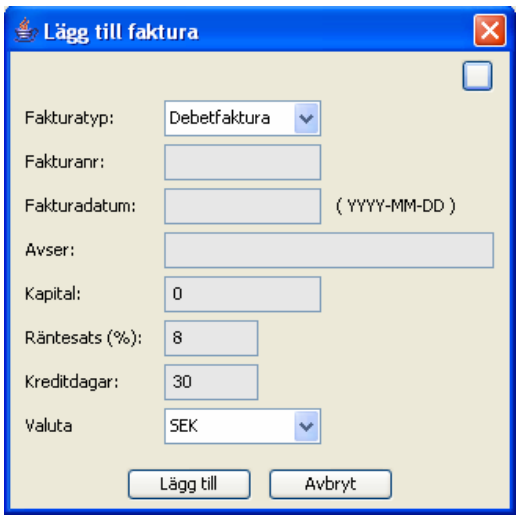
När användaren vill lägga till en gäldenär till ett befintligt ärende används formuläret LäggTillGäldenärGUI. Detta formulär skapas från panelen VisaÄrendePanel som beskrivs i avsnitt 5.3.3.2. Som inargument får då formuläret bland annat den ärendemodell som användaren vill registrera en gäldenär till. LäggTillGäldenärGUI är uppbyggt på samma sätt som den gäldenärstabb som beskrevs i avsnitt 5.3.4.1. Skillnaden är att det till ett befintligt ärende i programmet bara går att lägga till en gäldenär åt gången. Av den anledningen finns det ingen tabell i formuläret LäggTillÄrendeGUI. I Figur 5-11 visas en bild på formuläret.



Figur 5-11: Bild på LäggTillGäldenärGUI

5.3.4.4 LäggTillFakturaGUI

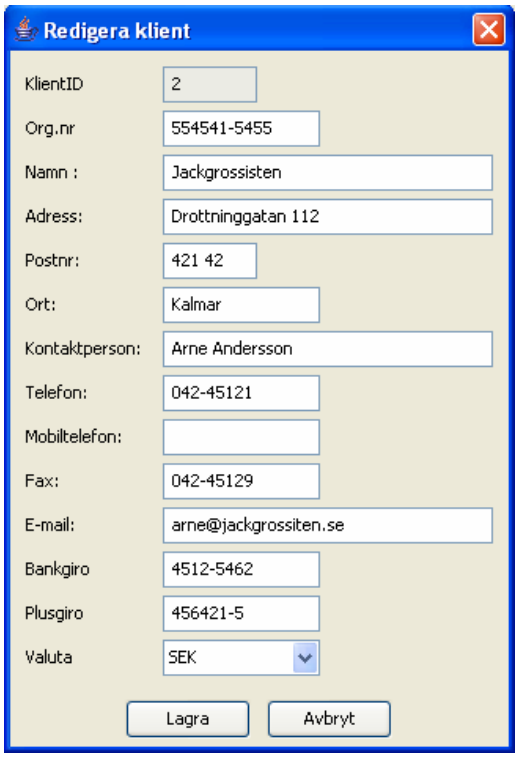
I programmet är det möjligt att registrera in en ny faktura till ett befintligt ärende. Formuläret skapas från panelen VisaÄrendePanel och får då den ärendemodellen som användaren vill registrera en faktura till som inargument. Det formuläret som används för att registrera in en ny faktura heter LäggTillFakturaGUI. Detta visas i Figur 5-12 och har samma textfält och comboboxar som fakturatabben som beskrevs i avsnitt 5.3.4.1. Även de kontroller som ser till att användaren matar in korrekt information om fakturan fungerar likadant. I



Figur 5-12: Bild på LäggTillFakturaGUI

5.3.4.5 RedigeraKlientGUI

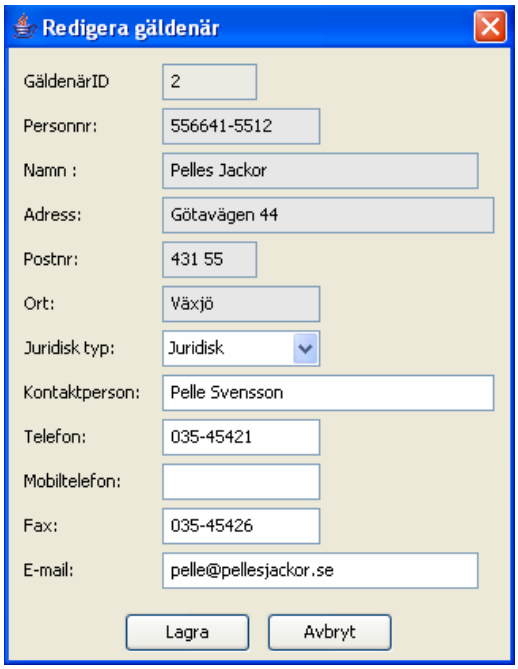
Ibland uppstår det tillfällen då användaren vill ändra information om en klient som redan är inregistrerad i databasen. För att kunna göra det används ett formulär som heter RedigeraKlientGUI. Det innehåller samma textrutor för att mata in information som klienttabben som är beskriven i avsnitt 5.3.4.1 samt knapparna Lagra och Avbryt. RedigeraKlientGUI skapas ifrån panelen VisaKlientPanel och skickar då med den klientmodell som ska redigeras som inargument. När formuläret skapas fylls de olika fälten med informationen från klientmodellen. En bild på RedigeraKlientGUI visas i Figur 5-13.



Figur 5-13: Bild på RedigeraKlientGUI

5.3.4.6 RedigeraGäldenärGUI

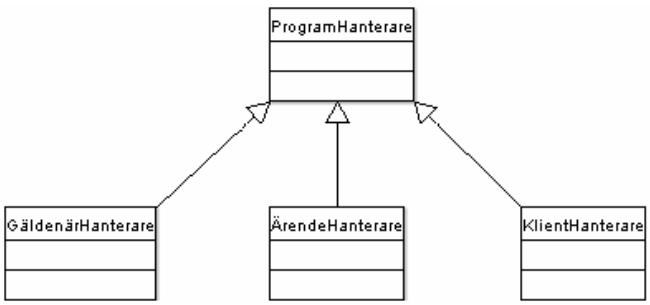
För att ändra informationen kring en gäldenär som redan är inregistrerad i databasen finns formuläret RedigeraKlientGUI. Formuläret innehåller samma textrutor för att mata in information som gäldenärtabben som beskrevs i avsnitt 5.3.4.1, detta visar Figur 5-14. Det har också två knappar, Lagra och Avbryt. Avbryt används för att stänga ner formuläret medan Lagra används för att utföra redigeringen av gäldenären. Formuläret skapas från panelen VisaGäldenärPanel som beskrevs i avsnitt 5.3.3.4. Panelen skickar då med den gäldenär som ska redigeras som inargument.



Figur 5-14: Bild på RedigeraGäldenärGUI

5.4 Hanterare

Hanterarens uppgift i programmet är att koppla samman de olika användargränssnitten samt att sköta kommunikationen mot databasklassen. Alla anrop som sker till databasklassen går via en hanterare. Detsamma gäller när ett användargränssnitt ska visa ett annat GUI. Anledningen till att koppla samma alla delarna i programmet med hjälp av hanterare är för att få en lösare koppling mellan dem. I programmet finns det fyra stycken hanterare. Dessa heter ProgramHanterare, ÄrendeHanterare, KlientHanterare och GäldenärHanterare. De olika hanterarna ansvarar för olika delar i programmet och beskrivs mer ingående i avsnitt 5.4.1 till och med 5.4.4. I varje avsnitt visas en lista över de GUI samt tjänster mot databasen som den aktuella hanteraren ansvarar över. Relationen mellan de olika hanterarna visas i Figur 5-15.



Figur 5-15: Klassdiagram över hanterarna

5.4.1 ProgramHanterare

Som Figur 5-15 visar är det programhanteraren som binder samman de olika hanterarna. Detta sker i och med att ärendehanteraren, gäldenärshanteraren och klienthanteraren ärver egenskaper ifrån programhanteraren. När programhanteraren instansieras skapas referenser till de övriga tre hanterarna. För att komma åt referenserna används metoderna hämtaÄrendeHanterare, hämtaGäldenärHanterare och så vidare. Programhanteraren har också en koppling till databasklassen via databasinterfacet DatabasAccess som beskrivs i avsnitt 4.2. I programhanteraren finns det metoder för att skapa och hämta programmets huvudfönster som heter InkassoGUI. Som beskrivits i avsnitt 5.3.1 är InkassoGUI bland annat uppbyggd av en tabbpanel av typen JTabbedPane. Programhanteraren har metoder för att lägga till respektive ta bort tabbar i tabbpanelen. I avsnitt 5.3 beskrevs att programmets användargränssnitt bland annat är uppdelat i paneler och formulär. Gemensamt för alla hanterare är att när ett formulär ska visas skapas en instans av det formuläret. Formuläret visas då automatiskt på skärmen. När däremot en panel ska visas skapas en instans av panelen som sedan placeras i en tabb i tabbpanelen. De användargränssnitt som programhanteraren ansvarar för är panelerna SökPanel och StartPanel.

5.4.2 ÄrendeHanterare

Ärendehanteraren har som uppgift att koppla samman alla delar i programmet som har med ärendet att göra. Detta innebär att visa de två formulären NyttÄrendeGUI och LäggTillFakturaGUI samt panelen VisaÄrendePanel. Ärendehanteraren ansvarar också för att lagra och söka efter ärenden samt att lagra och ta bort fakturor i databasen. Kopplingen till databasklassen får ärendehanteraren genom att den ärver egenskaper från programhanteraren.

5.4.3 KlientHanterare

För att koppla samman de olika användargränsnitten och databasfunktionerna som berör klienten används klienthanteraren. Klienthanteraren har metoder för att visa de olika användargränssnitten samt utföra sökning, lagring och redigering av klienter i databasen. För att anropa metoder i databasklassen används den referens som klienthanteraren ärver från programhanteraren. Klienthanteraren ansvarar också för att visa formulären NyKlientGUI och RedigeraKlientGUI samt panelen VisaKlientPanel.

5.4.4 GäldenärHanterare

Den hanterare som kopplar samman de delarna som rör gäldenären heter GäldenärHanterare. Den ansvarar för att visa panelen VisaGäldenärPanel samt formulären LäggTillGäldenärGUI

och RedigeraGäldenärGUI. Eftersom gäldenärhanteraren ärver egenskaper från programhanteraren ärver den också en referens till databasklassen. Denna referens används när gäldenärhanteraren ska söka, lagra och redigera gäldenärer i databasen.

6 Slutsatser

Jag har under detta examensarbete utvecklat en grund till ett inkassohanteringssystem. Ett program som ska utvecklas vidare för att sedan när det är klart kunna användas för att effektivisera Exactors ärendehantering. Grunden till inkassohanteringssystemet omfattade möjlighet för användaren att bland annat registrera, söka och redigera ett ärende och dess komponenter. Till programmet skapades också ett grafiskt användargränssnitt. Ett GUI där kraven var att det dels skulle vara lätt att överskåda men även lätt att använda.

Det har varit en mycket lärorik utmaning att göra examensarbetet. Kul att kunna bevisa för sig själv och andra att man genom de kunskaper man fått på dataingenjörsutbildningen kan utveckla ett program av denna storleksordning. Det har också varit extra kul att jag fått ha i stort sett fria händer för att utveckla designen av programmet.

Resultatet av examensarbetet blev över förväntan. Jag har bara fått positiva reaktioner kring programmet när jag presenterat det. Extra kul var det givetvis att uppdragsgivaren var väldigt nöjd och tror på mitt framtida arbete. Även uppsatsen blev bättre än vad jag vågat hoppas på. Eftersom jag inte skrivit några längre skrivarbeten på många år var förväntningarna på min skrivförmåga inte speciellt höga.

Innan examensarbetet började var jag osäker kring hur mycket tid som krävdes för det arbetet jag skulle utföra. Såhär i efterhand känner jag att tidsplaneringen var precis lagom. Jag är mycket nöjd med min egen arbetsinsats och känner att jag kan vara stolt över vad jag har presterat.

6.1 Erfarenhet och problem

Under projektets gång har en del problem uppstått. Problem som måste lösas och som givit en del erfarenheter. I detta avsnitt beskrivs de olika problemen samt hur lösningen blev.

När ett helt ärende lagras ner i databasen sparas det information som kopplar samman de olika tabellerna i ärendet. Ett exempel på detta är fältet KlientID i tabellen Ärende som innehåller id för den som är klient i ärendet. I avsnitt 5.2.1 beskrivs att när ny information sparas till en databastabell får primärnyckeln automatiskt ett värde. För att ta reda på vilket

värde KlientID fick i klienttabellen behövs en metod för detta. Till många databasmotorer finns det en metod som kollar vad den senaste lagrade tuplen fick för värde på primärnyckeln. Detta är dock inte fallet i SQLite om man ska anropa via JDBC. Eftersom värdet på primärnyckeln hela tiden ökar blev lösningen på problemet att skriva en metod som med hjälp av en SQL-fråga kollar vad det största värdet på primärnyckeln i en tabell är och returnera den.

Ett annat problem som uppstår vid lagring av ärende är när ärendet innehåller en klient eller gäldenär som redan finns registrerad i databasen. Vid dessa tillfällen ska inte klienten eller gäldenären lagras ner i databasen på nytt utan endast kopplas till ärendet. Av denna anledning måste en kontroll av klienten eller gäldenären ske. Lösningen på problemet blev att en ny klient eller gäldenär får sitt id satt till 0. På så sätt kan programmet identifiera om de redan finns lagrade i databasen.

I programmet uppstod ett problem när de olika valen i menyn skulle kopplas samman med de olika knapparna i GUI:t. Eftersom referenser till menyval och knappar ofta finns i olika klasser är det svårt att använda sig av dessa för att identifiera en händelse i programmet. Detta löstes genom att de olika komponenterna i GUI:t skickar ActionCommands när en händelse inträffar.

6.2 Framtida utökning och förbättring

Nu när grunden för inkassohanteringsprogrammet är skapad är det dags att börja se framåt till vad som händer efter examensarbetet. Som det är tänkt ska en framtida utveckling börja planeras efter det att examensarbetet är avslutat. De delar som ligger närmast till hands att skapa är:

- Utskrifter av olika typer av brev. Exempel på brev är påminnelse, inkassokrav, amorteringsplan med mera.
- Göra det möjligt att lägga till kostnader till ett ärende, såsom inkassokravsavgift med mera.
- Göra det möjligt att registrera inbetalningar som gäldenären gör till ett ärende. Detta innefattar hur programmet ska sköta avräkning på kostnad, ränta och kapital.
- Utökning till flera olika valutor där konverteringen mellan de olika valutorna sker automatiskt.

- Göra det möjligt att skapa en så kallad spärrlista där användaren kan se vilka ärenden som kräver en åtgärd.
- Inloggning till programmet för att öka säkerheten.
- Kryptering av den lagrade informationen i databasen

Referenser

[1] Datainspektionen ; http://www.datainspektionen.se ; 2005-12-11

[2] Cardell, Anders; Cardell, Jonas ; Kredit- och inkassohandboken ; ISBN 91-974178-1-5

[3] Klasson, Kristian ; Konsten att få betalt ; ISBN 91-39-00057-5

[4] Skatteverket ; http://www.skatteverket.se ; 2005-12-11

[5] Avtalslagen ; http://www.ekonomikonsulter.se/avtalslagen.asp ; 2005-12-11

[6] Alkefjärd, Peter ; Från faktura till betalt ; ISBN 91-631-4712-2

[7] Skansholm, Jan ; Java direkt med Swing ; ISBN 91-44-03843-7

[8] Eriksson, Hans-Erik ; Programutveckling med Java ; ISBN 91-44-00219-x

[9] SQLite home page ; http://www.sqlite.org ; 2005-12-11

A SkapaDatabas.java

```
import SQLite.JDBCDriver;
import SQLite.JDBC2y.*;
import java.sql.*;

public class SkapaDatabas {

    //Konstruktor
    SkapaDatabas( String dbNamn ) {
        databasNamn = dbNamn;
    }

    //Skapar SQL-frågorna
    public void förberedSQLFrågor() {
    try {
        sqlSkapaÄrendeTabell = uppkoppling.prepareStatement(
              "CREATE TABLE Arende ( ArendeID INTEGER PRIMARY KEY, " +
              "KlientID INTEGER, RegDatum VARCHAR, Status VARCHAR );" );

        sqlSkapaGäldenärTabell = uppkoppling.prepareStatement(
              "CREATE TABLE Galdenar ( GaldenarID INTEGER PRIMARY KEY, "+
              "RegDatum VARCHAR, PersonNr VARCHAR, Namn VARCHAR, " +
              "Adress VARCHAR, PostNr VARCHAR, " +
              "Ort VARCHAR, Kontaktperson VARCHAR, Telefon VARCHAR, " +
              "Mobiltelefon VARCHAR, Fax VARCHAR, Email VARCHAR, " +
              "JuridiskTyp VARCHAR, Valuta VARCHAR )" );

        sqlSkapaGäldenärListaTabell = uppkoppling.prepareStatement(
              "CREATE TABLE GaldenarLista ( GaldenarID INTEGER, " +
              "ArendeID INTEGER, PRIMARY KEY( GaldenarID, ArendeID ));");

        sqlSkapaKlientTabell = uppkoppling.prepareStatement(
              "CREATE TABLE Klient ( KlientID INTEGER PRIMARY KEY, " +
              "RegDatum VARCHAR, OrganisationsNr VARCHAR,Namn VARCHAR, "+
              "Adress VARCHAR, PostNr VARCHAR, Ort VARCHAR, " +
              "Kontaktperson VARCHAR, Telefon VARCHAR, " +
              "Mobiltelefon VARCHAR, Fax VARCHAR, Bankgiro VARCHAR, " +
              "Plusgiro VARCHAR, Email VARCHAR, Valuta VARCHAR );" );

        sqlSkapaFakturaTabell = uppkoppling.prepareStatement(
              "CREATE TABLE Faktura (FakturaID INTEGER PRIMARY KEY, " +
              "ArendeID INTEGER,FakturaTyp INTEGER,RegDatum VARCHAR, "+
              "FakturaNr VARCHAR, FakturanAvser VARCHAR, " +
              "FakturaDatum VARCHAR, UrsprungligtKapital FLOAT, " +
              "Referensranta FLOAT, Rantesats FLOAT, " +
              "AntalKreditdagar INTEGER, Valuta VARCHAR );" );
        }
        catch ( Exception exception ) {
              exception.printStackTrace();
        }
    }
```

```
//Skapar en uppkoppling till SQLitedatabasen
public void skapaUppkoppling() {
     try {
       //Registrerar JDBC-drivrutinen
       DriverManager.registerDriver( new JDBCDriver() );

       //Öppnar uppkopplingen mot SQLitedatabasen som finns under
       //C:\"databasNamn". Om inte databasen finns skapas den
       uppkoppling = new JDBCConnection( "jdbc:sqlite:/C:/" +
            "databasNamn, "" );
     }
     catch ( Exception e ) {
          e.printStackTrace();
     }
}

public void skapaTabellerna() {

     //Skapar uppkopplingen till databasen
     skapaUppkoppling();

     //Skapar SQL-frågorna
     förberedSQLFrågor();

     try {

       //Exekverar de olika SQL-frågorna och skapar då de olika
       //tabellerna
       sqlSkapaÄrendeTabell.executeUpdate();
       sqlSkapaGäldenärTabell.executeUpdate();
       sqlSkapaGäldenärListaTabell.executeUpdate();
       sqlSkapaKlientTabell.executeUpdate();
       sqlSkapaFakturaTabell.executeUpdate();

       //Stänger ner uppkopplingen
       uppkoppling.close();
     }
     catch ( Exception exception ) {
       exception.printStackTrace();
     }

public static void main(String[] args) {
     //Skapar en instans av klassen SkapaDatabas
     SkapaDatabas db = new SkapaDatabas( "inkassoDB.db" );

     //Skapar databasen
     db.skapaTabellerna();
}

//Referens till namnet på databasen
private String databasNamn;

//Referens till uppkopplingen mot databasen
private JDBCConnection uppkoppling;

//Referenser till SQL-frågorna
private PreparedStatement sqlSkapaÄrendeTabell;
private PreparedStatement sqlSkapaGäldenärTabell;
private PreparedStatement sqlSkapaGäldenärListaTabell;
private PreparedStatement sqlSkapaKlientTabell;
private PreparedStatement sqlSkapaFakturaTabell;
```

B Tabellbeskrivningar

* = Primärnyckel

B.1 Ärende

Fältnamn	Datatyp	Beskrivning
ÄrendeID*	INTEGER	Ärendets unika ID
KlientID	INTEGER	Klientens unika ID som sammankopplar ärendet med en specifik klient
RegDatum	TEXT	Datum då ärendet registrerades
Status	TEXT	Ärendets aktuella status

Tabell A-1: Tabellen Ärende

B.2 Klient

Fältnamn	Datatyp	Beskrivning
KlientID*	INTEGER	Klients unika ID
RegDatum	TEXT	Datum då klienten registrerades
OrganisationsNr	TEXT	Klientens organisationsnummer
Namn	TEXT	Klientens namn
Adress	TEXT	Klientens adress
PostNr	TEXT	Postnummer till klientens adress
Ort	TEXT	Ort till klientens adress
Kontaktperson	TEXT	Kontaktperson hos klienten
Telefon	TEXT	Klientens telefonnummer
Mobiltelefon	TEXT	Klientens mobiltelefonnummer
Fax	TEXT	Klientens faxnummer
Bankgiro	TEXT	Klientens bankgironummer
Plusgiro	TEXT	Klientens plusgironummer
Email	TEXT	Klientens e-mailadress
Valuta	TEXT	Klients valutakod (t ex SEK för svenska kronor)

Tabell A-2: Tabellen Klient

B.3 Gäldenär

Fältnamn	Datatyp	Beskrivning
GäldenärID*	INTEGER	Gäldenärens unika ID
RegDatum	TEXT	Datum då gäldenären registrerades
PersonNr	TEXT	Gäldenärens personnummer
Namn	TEXT	Gäldenärens namn
Adress	TEXT	Gäldenärens adress
PostNr	TEXT	Postnummer till gäldenärens adress
Ort	TEXT	Ort till gäldenärens adress
Kontaktperson	TEXT	Kontaktperson hos gäldenären
Telefon	TEXT	Gäldenärens telefonnummer
Mobiltelefon	TEXT	Gäldenärens mobiltelefonnummer
Fax	TEXT	Gäldenärens faxnummer
Email	TEXT	Gäldenärens e-mailadress
JuridiskTyp	TEXT	Juridisk typ på gäldenären

Tabell A-3: Tabellen Gäldenär

B.4 GäldenärLista

Fältnamn	Datatyp	Beskrivning
GäldenärID*	INTEGER	Gäldenärens unika ID
ÄrendeID*	INTEGER	Ärendets unika ID

Tabell A-4: Tabellen GäldenärLista

B.5 Faktura

Fältnamn	Datatyp	Beskrivning
FakturaID*	INTEGER	Fakturans unika ID
ÄrendeID	INTEGER	Ärendets unika ID som sammankopplar fakturan med det aktuella ärendet.
RegDatum	TEXT	Datum då fakturan registrerades
FakturaTyp	TEXT	Fakturans typ
FakturaNr	TEXT	Fakturans unika nummer hos klienten
FakturanAvser	TEXT	Information om vad fakturan avser
UrsprungligtKapital	FLOAT	Fakturans ursprungliga kapital
Referensränta	FLOAT	Den aktuella referensräntan
Räntesats	FLOAT	Avtalad räntesats på fakturan
AntalKreditdagar	INTEGER	Antalet kreditdagar på fakturan
Valuta	TEXT	Fakturans valutakod(t ex SEK för svenska kronor)

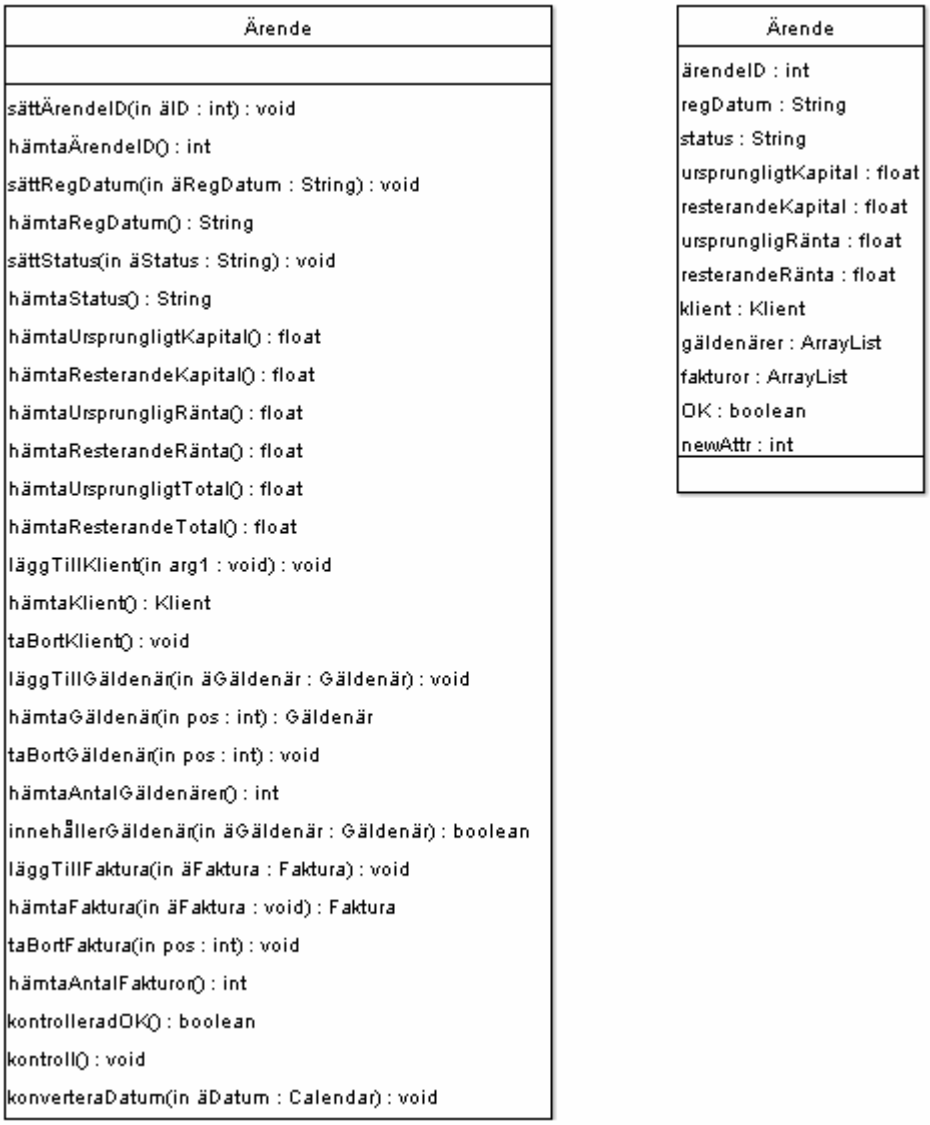
Tabell A-5: Tabellen Faktura

C DatabasAccess.java

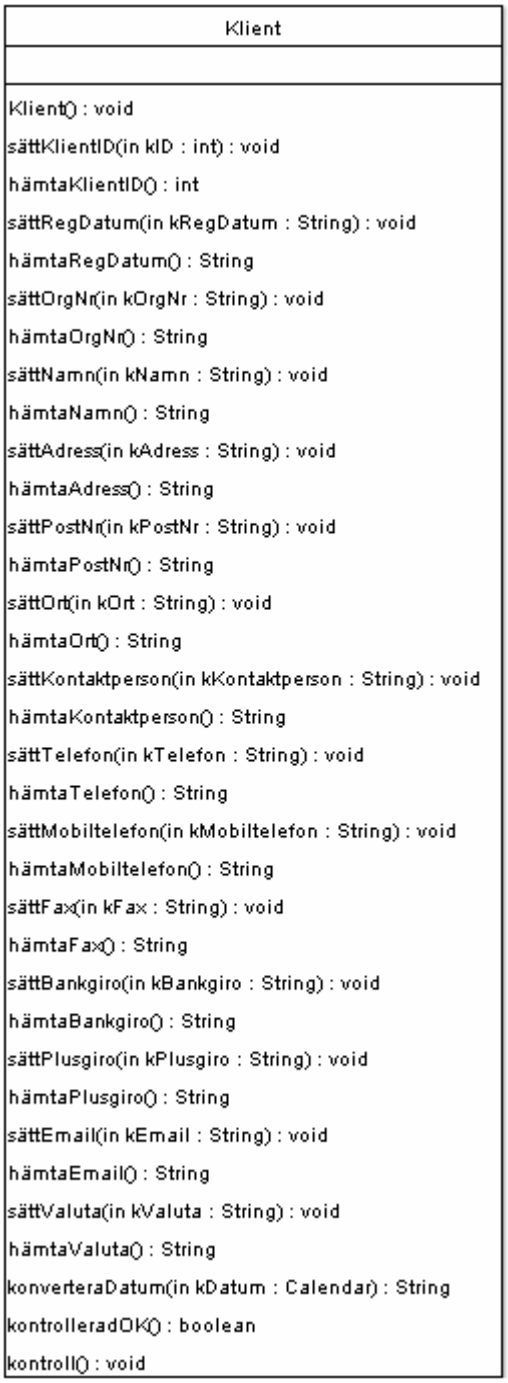
```
import java.util.ArrayList;

public interface DatabasAccess {
      //Sparar ner gäldenären i databasen
     public boolean läggTillGäldenär( Gäldenär gäldenär, int ärendeID );

     //Tar bort en gäldenär från ett ärende databasen
     public boolean taBortGäldenär( int gäldenärID, int ärendeID );

     //Uppdaterar en befintlig gäldenär i databasen
     public boolean uppdateraGäldenär( Gäldenär gäldenär );

     //Hämtar gäldenären med aktuellt gäldenärID från databasen
     public Gäldenär hämtaGäldenär( int gäldenärID );

     //Söker med hjälp av sökIndex och sökVärde fram gäldenärer från
     //databasen och returnerar träffarna i en ArrayList.
     public ArrayList sökGäldenär( int sökIndex, String sökVärde );

     //Sparar ner klienten i databasen
     public boolean sparaKlient( Klient klient );

     //Uppdaterar en befintlig gäldenär i databasen
     public boolean uppdateraKlient( Klient klient );

     //Hämtar klienten med aktuellt klientID från databasen
     public Klient hämtaKlient( int klientID );

     //Söker med hjälp av sökIndex och sökVärde fram klienter från
     //databasen och returnerar träffarna i en ArrayList.
     public ArrayList sökKlient( int sökIndex, String sökVärde );

     //Sparar ner fakturan i databasen
     public boolean sparaFaktura ( Faktura faktura, int ärendeID );


     //Hämtar fakturan med aktuellt fakturaID från databasen
     public Faktura hämtaFaktura( int fakturaID );

     //Tar bort fakturan med aktuellt fakturaID från databasen
     public boolean taBortFaktura( int fakturaID );

     //Sparar ner ärendet i databasen
     public boolean sparaÄrende( Ärende ärende );

     //Uppdaterar ett befintligt ärende i databasen
     public boolean uppdateraÄrende( Ärende ärende );

     //Hämtar ärendet med aktuellt ärendeID från databasen
     public Ärende hämtaÄrende( int ärendeID );

     //Söker med hjälp av sökIndex och sökVärde fram ärenden från
     //databasen och returnerar träffarna i en ArrayList.
     public ArrayList sökÄrende( int sökIndex, String sökVärde );
}
```

D Modellbeskrivningar

D.1 Ärende

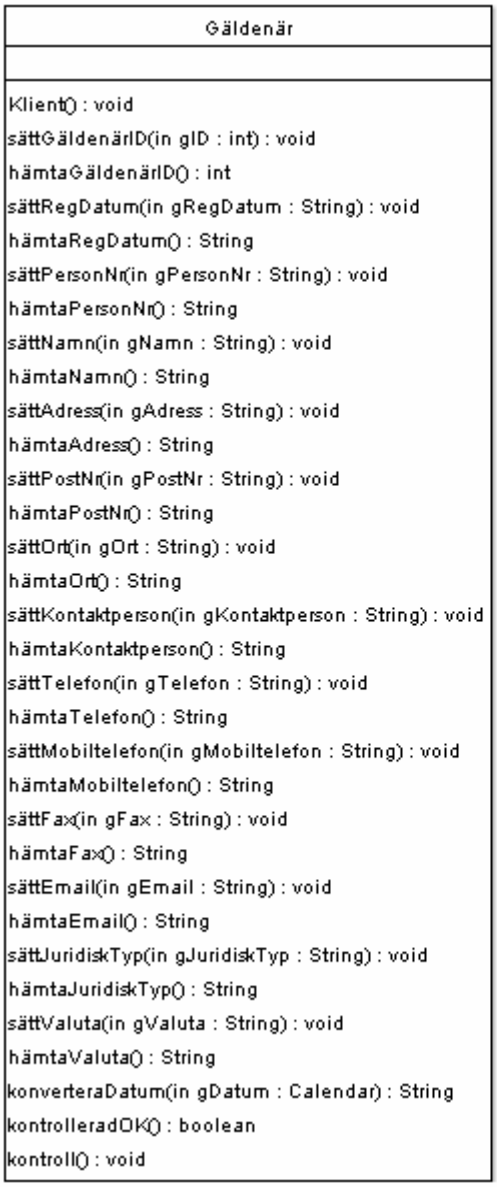


D.2 Klient


Klient
Klient() : void
sättKlientID(in kID : int) : void
hämtaKlientID() : int
sättRegDatum(in kRegDatum : String) : void
hämtaRegDatum() : String
sättOrgNr(in kOrgNr : String) : void
hämtaOrgNr() : String
sättNamn(in kNamn : String) : void
hämtaNamn() : String
sättAdress(in kAdress : String) : void
hämtaAdress() : String
sättPostNr(in kPostNr : String) : void
hämtaPostNr() : String
sättOrt(in kOrt : String) : void
hämtaOrt() : String
sättKontaktperson(in kKontaktperson : String) : void
hämtaKontaktperson() : String
sättTelefon(in kTelefon : String) : void
hämtaTelefon() : String
sättMobiltelefon(in kMobiltelefon : String) : void
hämtaMobiltelefon() : String
sättFax(in kFax : String) : void
hämtaFax() : String
sättBankgiro(in kBankgiro : String) : void
hämtaBankgiro() : String
sättPlusgiro(in kPlusgiro : String) : void
hämtaPlusgiro() : String
sättEmail(in kEmail : String) : void
hämtaEmail() : String
sättValuta(in kValuta : String) : void
hämtaValuta() : String
konverteraDatum(in kDatum : Calendar) : String
kontrolleradOK() : boolean
kontroll() : void

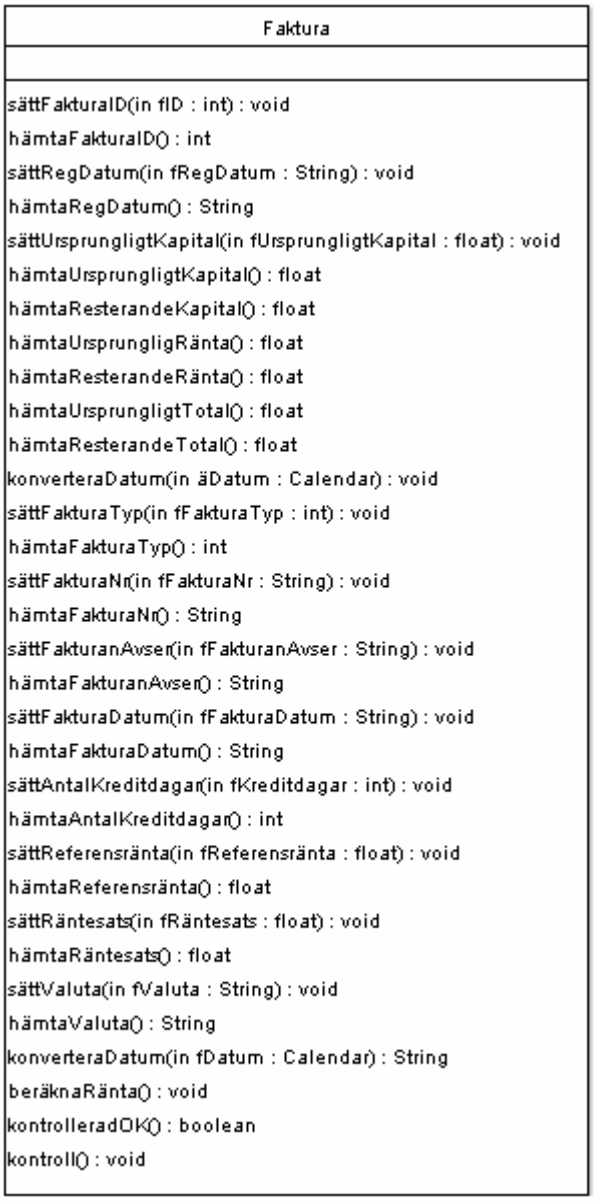

Klient
klientID : int
regDatum : String
orgNr : String
namn : String
adress : String
postNr : String
ort : String
kontaktperson : String
telefon : String
mobiltelefon : String
fax : String
bankgiro : String
plusgiro : String
email : String
valuta : String
OK : boolean

D.3 Gäldenär


Gäldenär
Klient() : void
sättGäldenärID(in gID : int) : void
hämtaGäldenärID() : int
sättRegDatum(in gRegDatum : String) : void
hämtaRegDatum() : String
sättPersonNr(in gPersonNr : String) : void
hämtaPersonNr() : String
sättNamn(in gNamn : String) : void
hämtaNamn() : String
sättAdress(in gAdress : String) : void
hämtaAdress() : String
sättPostNr(in gPostNr : String) : void
hämtaPostNr() : String
sättOrt(in gOrt : String) : void
hämtaOrt() : String
sättKontaktperson(in gKontaktperson : String) : void
hämtaKontaktperson() : String
sättTelefon(in gTelefon : String) : void
hämtaTelefon() : String
sättMobiltelefon(in gMobiltelefon : String) : void
hämtaMobiltelefon() : String
sättFax(in gFax : String) : void
hämtaFax() : String
sättEmail(in gEmail : String) : void
hämtaEmail() : String
sättJuridiskTyp(in gJuridiskTyp : String) : void
hämtaJuridiskTyp() : String
sättValuta(in gValuta : String) : void
hämtaValuta() : String
konverteraDatum(in gDatum : Calendar) : String
kontrolleradOK() : boolean
kontroll() : void


Gäldenär
gäldenärID : int
regDatum : String
personNr : String
namn : String
adress : String
postNr : String
ort : String
kontaktperson : String
telefon : String
mobiltelefon : String
fax : String
email : String
juridiskTyp : String
valuta : String
OK : boolean

D.4 Faktura


Faktura
sättFakturaID(in fID : int) : void
hämtaFakturaID() : int
sättRegDatum(in fRegDatum : String) : void
hämtaRegDatum() : String
sättUrsprungligtKapital(in fUrsprungligtKapital : float) : void
hämtaUrsprungligtKapital() : float
hämtaResterandeKapital() : float
hämtaUrsprungligRänta() : float
hämtaResterandeRänta() : float
hämtaUrsprungligtTotal() : float
hämtaResterandeTotal() : float
konverteraDatum(in äDatum : Calendar) : void
sättFakturaTyp(in fFakturaTyp : int) : void
hämtaFakturaTyp() : int
sättFakturaNr(in fFakturaNr : String) : void
hämtaFakturaNr() : String
sättFakturanAvser(in fFakturanAvser : String) : void
hämtaFakturanAvser() : String
sättFakturaDatum(in fFakturaDatum : String) : void
hämtaFakturaDatum() : String
sättAntalKreditdagar(in fKreditdagar : int) : void
hämtaAntalKreditdagar() : int
sättReferensränta(in fReferensränta : float) : void
hämtaReferensränta() : float
sättRäntesats(in fRäntesats : float) : void
hämtaRäntesats() : float
sättValuta(in fValuta : String) : void
hämtaValuta() : String
konverteraDatum(in fDatum : Calendar) : String
beräknaRänta() : void
kontrolleradOK() : boolean
kontroll() : void


Faktura
fakturaID : int
regDatum : String
fakturaTyp : int
fakturaNr : String
fakturanAvser : String
fakturaDatum : String
ursprungligtKapital : float
resterandeKapital : float
ursprungligRänta : float
resterandeRänta : float
kreditdagar : int
referensränta : float
räntesats : float
valuta : String
OK : boolean

E Metoder för att skapa komponenter i Standardpanel

```
//Skapar en Etikett med texten etikettNamn med font. Etiketten placeras på
//panelen med hjälp av instruktion.
public void skapaEtikett(String etikettNamn, String instruktion, Font font)

//Skapar ett tomt textfält som heter FältNamn med storleken dimension.
//Textrutan placeras på panelen med hjälp av instruktion och lagras i
//komponentHashMap.
public void skapaTextFält( String fältNamn, String instruktion,
                                Dimension dimension )

//Skapar en tom textruta som heter textFältNamn med storleken dimension.
//Textrutan placeras på panelen med hjälp av instruktion och lagras i
//komponentHashMap.
public void skapaTextRuta( String textFältNamn, String instruktion,
                                Dimension dimension )


//Skapar en tom combobox som heter comboBoxNamn med storleken dimension och
//fylls med värde. Comboboxen placeras på panelen med hjälp av instruktion
//och lagras i komponentHashMap.
public void skapaComboBox( String comboBoxNamn, String[] värde,
                                String instruktion, Dimension dimension )


//Skapar en tom tabell som heter tabellNamn med storleken dimension och har
//tabellmodellen tabellModellNamn. Tabellen har kolumnerna kolumnNamn och
//placeras på panelen med hjälp av instruktion och lagras i komponentHashMa
public void skapaTabell( String tabellNamn, String tabellModellNamn,
           String[] kolumnNamn, String instruktion, Dimension dimension )


//Skapar en knapp med namnet och texten knappNamn som har storleken
//dimension. Knappen kopplas till ActionListener lyssnare, placeras på
//panelen med hjälp av instruktion och lagras i komponentHashMap.
public void skapaKnapp( String knappNamn, String instruktion,
                     Dimension dimension,  ActionListener lyssnare )


//Skapar en knapp med namnet knappNamn och ikonen ikon som har storleken
//dimension. Knappen kopplas till ActionListener lyssnare, placeras på
//panelen med hjälp av instruktion och lagras i komponentHashMap.
public void skapaKnapp(String knappNamn, ImageIcon ikon,Stringinstruktion,
                     Dimension dimension, ActionListener lyssnare )


//Skapar en checkbox med namnet checkBoxNamn och testen titel. Checkboxen
//placeras på panelen med hjälp av instruktion och lagras i
//komponentHashMap
public void skapaCheckBox( String checkBoxNamn, String titel,
                     String instruktion, ActionListener lyssnare )
```