Datavetenskap

Basam Odisho

Saiwan Kamber

Utveckling av en bilskrotsdatabas

Examensarbete, C-nivå

2005:03

Utveckling av en bilskrotsdatabas

Basam Odisho

Saiwan Kamber

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.

Basam Odisho

Saiwan Kamber

Godkänd, 2005-01-14

Handledare: Katarina Asplund

Examinator: Martin Blom

Sammanfattning

Vi har arbetat på uppdrag av Kristinehamns Nya Bildemontering. Målet med vårt arbete var att skapa ett dataprogram med en grafisk lösning som samverkar lokalt med en databas, där användaren kan söka och manipulera data som består av bildemonteringens inventering på ett säkert och effektivt sätt. Resultatet av arbetet är en fullt fungerande prototyp som skall vidareutvecklas efter slutförande av denna rapport. Med hänsyn till att detta program skall användas kommersiellt är inte dess kod medtaget i denna rapport.

Development of a scrapyard database

Abstract

We have developed a program on behalf of Kristinehamns Nya Bildemontering. The purpose of this project was to develop a computer program with a grafical solution that interacts with a local database, which offers the user the oppurtunity to search and manipulate the company’s inventory in a secure and efficient manner. The result of this project is a fully functional prototype that will be further developed on the completion of this documentation.

With consideration that the program will be used commercialy, none of the code is posted in this documentation.

Innehållsförteckning

Figurförteckning

1 Inledning

Det här examensarbetet är utfört för Kristinehamns Nya Bildemonterings räkning. Detta företag har specialiserat sig på demontering och försäljningen av bildelar. Naturligtvis handlar det om en stor mängd bildelar som måste arkiveras. I dagsläget får uppdragsgivaren och hans anställda förlita sig på sitt minne, pappersarbete och arkivering som måste utföras för hand.

Lagerhantering är idag ett stort och tidskrävande arbete där uppdragsgivaren måste kombinera papper, pärmar och flera olika administrativa datorprogram såsom Excel och Word med varandra. Uppdragsgivaren är angelägen om att hans företag ska vara serviceinriktat, där kunden snabbt kan få den information denne söker, vilket är svårt i dagsläget. Då en bildel har sålts måste alla listor och program uppdateras, alla anställda bli informerade så att inte en kund blir lovad en bildel som inte längre finns i lager.

1.1 Syfte

Syftet med detta arbete är att effektivisera företagets lagerhantering genom att sammanfoga alla deras arkiv och datorverktyg under ett och samma datorprogram, som vi implementerar. Vårt datorprogram är en grafisk lösning som samverkar med en databas där användaren kan söka och manipulera data på ett säkert och effektivt sätt. Detta leder till att de ovannämnda problemen försvinner och uppdragsgivaren kan koncentrera sig på det administrativa arbetet.

1.2 Uppsatsens disposition

I kapitel 2 ges en kort historik och en beskrivning av demonteringsprocessen i en bilskrot. Detta följs av kapitel 3 vilket behandlar kravspecifikationen som tagits fram och fastställts i samarbete med uppdragsgivaren. I kapitel 4 och 5 beskriver vi designen för databasen och användargränssnitten för vårt program, och i kapitel 6 och 7 behandlas de tekniska aspekterna bakom databasen och användargränssnittet i detalj. I kapitel 8 behandlas kopplingsprocessen mellan databasen som är skapad i MySQL Server och användargränssnittet som är implementerat i språket Java. Examensarbetet avslutas med kapitel 9 där erfarenheterna och slutsatserna av arbetet är nedtecknade.

2 Bakgrundsfakta

I delavsnitt 2.1 ges en kort historik om bilskrotars uppkomst och delavsnitt 2.2 tar upp processen då en bil tas emot av en bilskrot. Delavsnitt 2.3 och 2.4 förklarar demonterings- respektive fragmenteringsprocessen.

2.1 Kort historik

På 1920-talet insåg man att det fanns ett behov av att förvara och hantera alla bilar som hade tagits ur bruk. Detta löste man genom att skapa bilskrotar där bilarna demonterades och deras användbara komponenter såldes som reservdelar. Fram till början av 1970-talet, när den första fragmenteringsanläggningen - en återvinningsanläggning av metall - togs i bruk, skedde all sortering av metaller och annat material för hand innan de tomma karosserna såldes som plåtskrot. Detta gör bildemontering till en av Sveriges äldsta återvinningsverksamheter.

På senare år har samhället blivit allt mer miljömedvetet och i samband med det har nyare och hårdare miljökrav börjat ställas på bilskrotsindustrin. Detta har lett till att alla bilskrotar enligt lag måste auktoriseras av länsstyrelsen och detta kan endast göras då alla miljökrav genomförts. För att garantera att alla miljökraven efterföljs måste alla bilskrotar få sina auktorisationer förnyade vart femte år. Som ytterligare försäkring på att miljökraven följs måste alla bilskrotar ha kontrakt med en fragmenterare som specialiserat sig på att kassera metall. Allt detta har medfört att standarden på bilskrotarna har höjts radikalt på senare år, men det har också medfört att många bilskrotar inte uppfyllt de nya hårda kraven och tvingats att lägga ner sina verksamheter [1].

2.2 Bilen kommer in

Skrotningsprocessen börjar med att skroten får in ett fordon, antingen genom att dess ägare lämnar in den eller genom att bilskroten bärgar den själv mot en avgift. Innan skrotningsprocessen påbörjas måste bilskroten ha ägarens medgivande, vilket fås genom att ägaren skriver över alla rättigheter till bilskroten. När bilskroten väl har bilen i sin ägo blir nästa steg att avregistrera fordonet, något som numera sker via datorn. Bilskroten kopplar upp sig mot Bilvision AB[1] –en av Vägverket auktoriserad återförmedlare – och avregistrerar fordonet som sedan skrotas.

Med avregistrering menar vi att fordonet i fråga officiellt tas ur bruk och inte längre får köras. Med skrotning menar vi att det avregistrerade fordonet fysiskt demoleras och skroten inte längre kan demontera säljbara delar ifrån det. Demonteringsprocessen tar vi en närmare titt på i nästa delavsnitt.

2.3 Demonteringsprocess

Demonteringsprocessen består av tre moment.

1. Moment ett består av att skroten tömmer fordonet på alla vätskor och demonterar miljöfarliga komponenter såsom katalysatorer, oljefilter, airbags, däck, batterier etc. De demonterade komponenterna måste förvaras enligt föreskrivna miljöstadgar.

2. Moment två består av att skroten demonterar de mindre komponenterna som ska säljas vidare som reservdelar. De större komponenterna såsom t.ex. motorblock demonteras inte utan förvaras i karossen till dess komponenterna efterfrågas.

3. Moment tre består av att alla fordon som har tömts på alla säljbara komponenter skickas till fragmentering.

[1] http://www.bilvision.se

3 Kravspecifikation

Kravspecifikationen för programmet togs fram i samarbete med företaget, där vi och uppdragsgivaren presenterade olika idéer för varandra. Slutligen fastställde vi att programmet skulle bestå av två delar; ett användargränssnitt och en databas. Programmet skall användas på en PC med operativsystemet Windows XP för att vara kompatibelt med arbetsgivarens arbetsdator. Specifikationerna för databasen och gränssnittet tas upp i delavsnitt 3.1 respektive 3.2.

3.1 Databas

- Databasen skall implementeras i programmet My-SQL [2] för Windows.
- Databasen skall bestå av fyra delar:
  1. Bilregister: En databas över avregistrerade och skrotade bilar som finns eller har funnits i företagets ägo.
  2. Bildel: En databas över alla reservdelar som finns på lager.
  3. Inköpslista: En databas över alla slutsålda reservdelar, som alltid skall vara synlig för användaren (implementeras i mån av tid).
  4. Däck: En databas över alla däck som finns på lager.

- Bildel, Bilregister och Däck skall ha följande egenskaper:
  1. Inmatningsfunktion: skall kunna hantera inmatning av ett objekt vid varje anrop.
  2. Borttagningsfunktion: skall kunna hantera borttagning av ett objekt vid varje anrop.
  3. Sökfunktion: skall kunna söka ett objekt åt gången.
  4. Utskriftsfunktion: skall kunna skriva ut innehållet ur tabellen.

3.2 Användargränssnitt (GUI)

- GUI:t skall implementeras i språket Java p.g.a. dess plattformsoberoende och dess stöd för databaser.

- GUI:t skall vara användarvänligt och användarsäkert, genom att användarens frihet begränsas m.h.a. rullgardiner etc.

- GUI:t skall bestå av fem delar:
  1. Bilregister: Ett GUI för databasen över avregistrerade och skrotade bilar som finns eller har funnits i företagets ägo.
  2. Bildelar: Ett GUI för databasen över alla reservdelar som finns på lager.
  3. Inköpslista: Ett GUI för databasen över alla slutsålda reservdelar, som alltid skall vara synlig för användaren (implementeras i mån av tid).
  4. Däck: Ett GUI för databasen över alla däck som finns på lager.
  5. Skal: Ett GUI som inkapslar alla ovanstående punkter.

- GUI för bildelar, bilregister och däck skall ha följande egenskaper:
  1. Inmatningsfunktion som skall kunna hantera inmatning av ett objekt vid varje anrop till databasen.
  2. Borttagningsfunktion som skall kunna hantera borttagning av ett objekt vid varje anrop till databasen.
  3. Sökfunktion som skall kunna söka ett objekt åt gången vid anrop till databasen.
  4. Tömfunktion som ska kunna nollställa alla fält på sidan.
  5. Uppdaterafunktion som ska kunna hämta de senaste värdena ur databasen.
  6. Visafunktion som ska kunna hämta de senaste värdena ur databasen.

- Skalet ska ha följande egenskaper:
  1. Skalet skall ge användaren möjligheten att navigera mellan de olika registren via en huvudmeny.
  2. Användaren ska kunna avsluta programmet via huvudmenyn.

4 Databasdesign

I detta kapitel kommer vi att ta en närmare titt på databasdesignen. Som tidigare nämnts i kapitel 3.1 så är databasen indelad i fyra delar; *Bilregister, Däck, Bildel* och *Inköpslista*. Designen för dessa delar tas upp i tur och ordning i delavsnitten 4.1- 4.4. Kapitel fyra avslutas med en kommentar i delavsnitt 4.5.

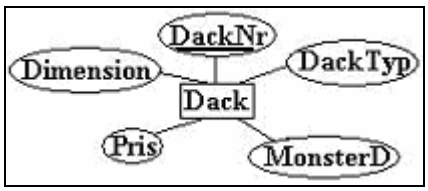
4.1 Bilregister

Databasen bilregister består av de två entiteterna *avregistrerade bilar* (se figur 4.1) och *skrotade bilar* (se bilaga A). *Avregistrerade bilar* innehåller de bilar som har avregistrerats av bilskroten, medan *skrotade bilar* innehåller de bilar som har avregistrerats och sedan skrotats av bilskroten. Detta innebär att innehållet i *skrotade bilar* består av de borttagna objekten ur tabellen *avregistrerade bilar*. Entiteten *avregistrerade bilar* har egenskaperna som kan ses i figur 4.1 och har de fyra operationerna; inmatning, borttagning, sökning och utskrift, medan tabellen *skrotade bilar* har endast operationen utskrift. Detta p.g.a. att de skrotade bilarna inte längre existerar och tabellen endast uppfyller en statistisk funktion.

Figur 4.1 E/R Modell för entiteten Avregistrerade bilar

4.2 Däck

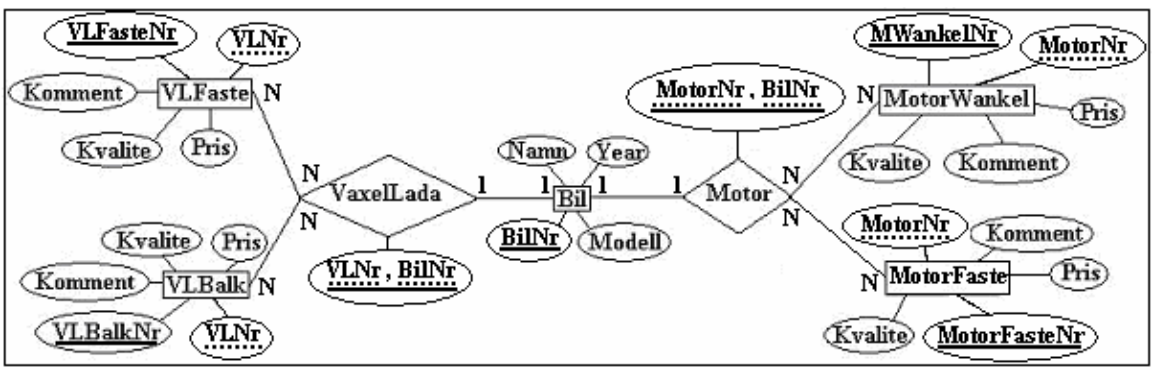
Databasen *Däck* består av en entitet (se figur 4.2) som innehåller alla säljbara sommar-, vinter- och dubbdäck som finns i bilskrotens lager. Entiteten *Dack* har de egenskaper som kan ses i figur 4.2 och har de fyra operationerna; inmatning, borttagning, sökning och utskrift. Till skillnad från databasen *bilregister* behövs inte något statistiskt register, detta p.g.a. överflödet av alla de typer av däck som kommer in till bilskroten. När ett däck är sålt så tas den bort ur databasen permanent.



Figur 4.2 E/R-modell för entiteten Dack

4.3 Bildel

Databasen *Bildel* innehåller alla bildelar som finns inne i bilskrotens lager. Dessa bildelar är grupperade i kategorier såsom motor och inredning. Som exempel kan nämnas att ett motorblock och en motorwankel är två bildelar som tillhör en motor som i sin tur tillhör en bil (se i figur 4.3). Därför är det naturligt att även i databasen kategorisera alla bildelar på detta sätt. *Bildel* har egenskaperna som kan ses i figur 4.3 och har de fyra operationerna; inmatning, borttagning, sökning och utskrift. När det sista exemplaret av en bildel är borttaget ur *Bildel* noteras detta i databasen *Inköpslista*.



Figur 4.3 E/R-modell för databasen Bildel

4.4 Inköpslista

Inköpslista är en databas över alla bildelar som är slutsålda och inte längre finns i bilskrotens lager. Detta innebär att när det sista objektet av en specifik typ i databasen *Bildelar* tas bort så noteras detta i *Inköpslistan* för att uppmärksamma användaren om detta. När bilskroten åter fått in det tidigare noterat slutsålda objektet i databasen *Bildelar*, ska denna notis försvinna ur *Inköpslistan*. Detta innebär att databasen *Inköpslista* endast har två operationer; inmatning och borttagning.

4.5 Mappning av E/R-modellen till relationsdatamodellen

Alla ovan beskrivna entiteter mappas till tabeller i en relationsdatabas, dessa beskrivs i bilaga A.

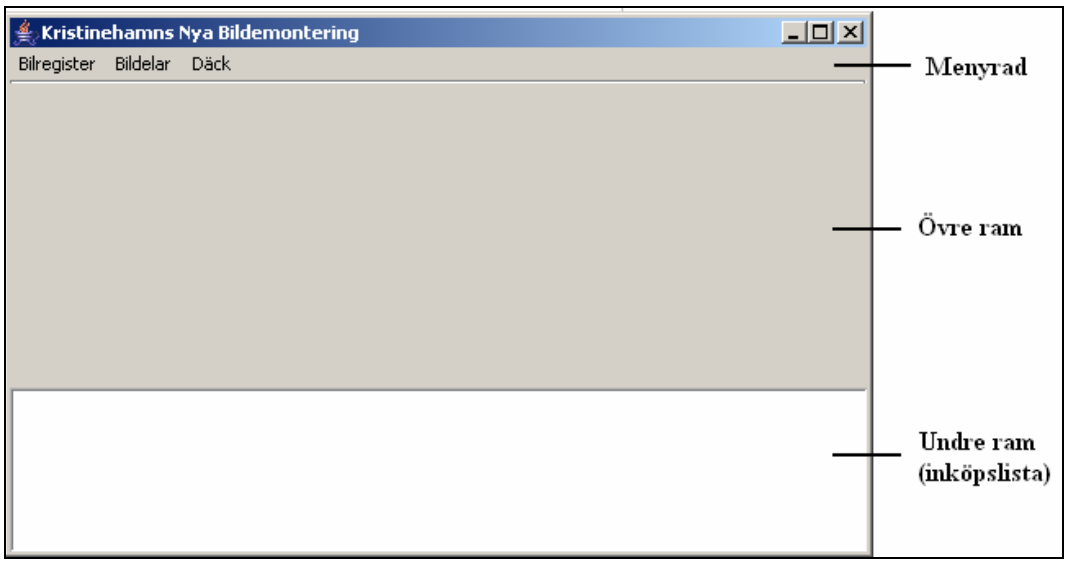
5 Grafisk användargränssnittsdesign (GUI-design)

I detta kapitel kommer vi att ta en närmare titt på GUI-designen som är indelad i fyra delar; *Skal, Bilregister, Bildelar och Däck*. Designen för dessa delar tas upp i tur och ordning i delavsnitten 5.1-5.4 .

5.1 Skal

När programmet har startats så visas det ett skal, som består av en ram innehållande en menyrad och två inre ramar (se figur 5.1). Menyraden innehåller tre menyer; *Bilregister, Bildelar* och *Däck*. De ovannämnda menyerna ger användaren möjligheten att navigera mellan de olika databaserna som togs upp i kapitel 4. De har även undermenyer som ger användaren tillgång till de olika databasernas funktioner, vilka beskrivs i de kommande delavsnitten 5.2-5.4.

De två inre ramarna, vilka är helt oberoende, är placerade under varandra. Den övre ramen innehåller alla fönster vilka öppnas då användaren väljer en funktion ur de ovannämnda menyerna, medan den undre ramen alltid är synlig och innehåller resultatet från databasen *Inköpslista* som nämndes i kapitel 4.4.



Figur 5.1 Bild över Skalet

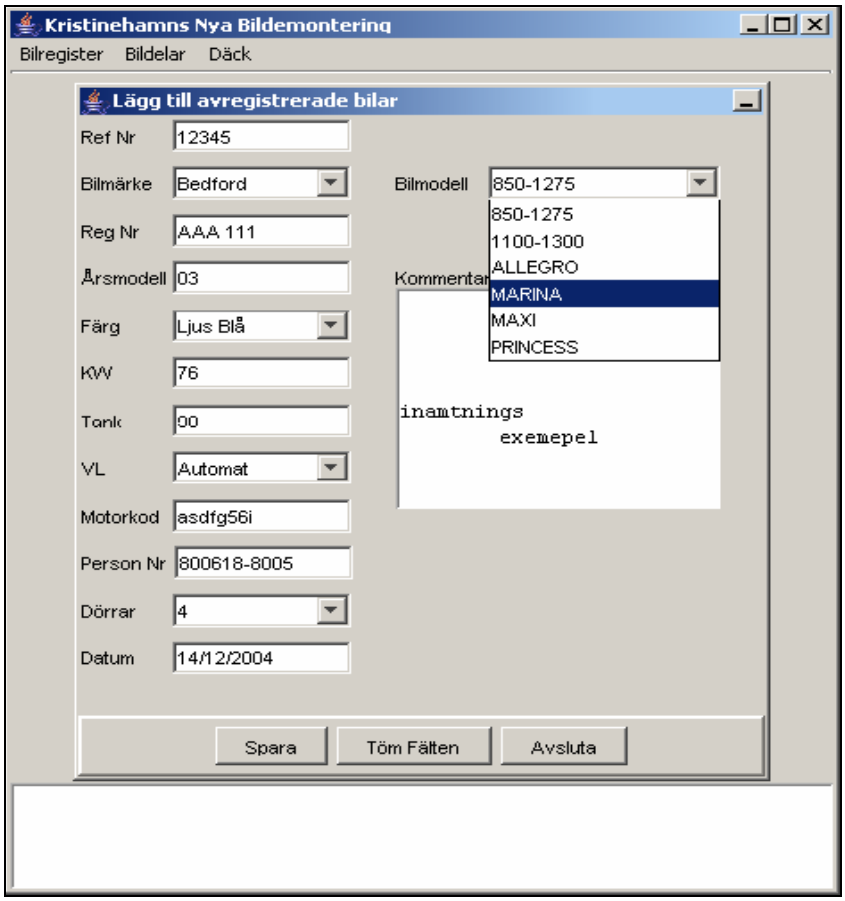
5.2 Bilregister

Menyn *Bilregister* innehåller en undermeny som består av tre alternativ; *Avregistrerade bilar, Skrotade bilar* och *Exit.* Alternativet *Exit* avslutar hela programmet och är alltid tillgängligt för användaren medan *Avregistrerade bilar* och *Skrotade bilar* är mer omfattande och beskrivs i närmare detalj i de kommande delavsnitten 5.2.1 och 5.2.2.

5.2.1 Avregistrerade bilar

Avregistrerade bilar innehåller en undermeny bestående av de tre alternativen *Lägg till, Ta bort/Sök* och *Visa* som ger användaren möjligheten att manipulera databasen *Bilregister*.

Då användaren valt något av de ovanstående alternativen öppnas en ny sida i den *övre ramen* som nämndes i delavsnitt 5.1. Då den nya sidan är aktiv kan inte samma menyalternativ väljas igen förrän användaren stänger ner den aktuella sidan.



Figur 5.2 Bild över Avregistrerade bilar - Lägg till

1. Alternativet *Lägg till*: Användaren har möjligheten att lägga till en ny bil i databasen *Bilregister*. Den nya sidan innehåller rullgardiner med fördefinierade värden och inmatningsrutor där användaren får mata in de värden som krävs för att lägga till en ny bil (se figur 5.2).

   Längst ner på sidan finns tre knappar:
   - *Spara* ger användaren möjligheten att spara de angivna värdena i databasen *Bilregister*.
   - *Töm fälten* ger användaren möjligheten att nollställa alla värden på sidan
   - *Avsluta* ger användaren möjligheten att stänga ner sidan.

2. Alternativet *Ta bort/Sök*: Användaren har möjligheten att söka eller ta bort en bil från databasen *Bilregister*. Den nya sidan innehåller rullgardiner med fördefinierade värden och inmatningsrutor där användaren får mata in de värden som krävs för att söka efter en bil i databasen. Då användaren fått sitt resultat på sökningen ges möjligheten att ta bort en bil åt gången från databasen genom att markera den rad som ska tas bort och trycka på knappen *Ta bort*.

   Längst ner på sidan finns fyra knappar:
   - *Sök* ger användaren möjligheten att söka efter bilar i databasen *Bilregister*.
   - *Ta bort* ger användaren möjligheten att ta bort den markerade bilen i sökresultatet från databasen *Bilregister*.
   - *Uppdatera* ger användaren möjligheten att nollställa alla värden på sidan och uppdaterar utskriften.
   - *Avsluta* ger användaren möjligheten att stänga ner sidan.

3. Alternativet *Visa*: Användaren har möjligheten att skriva ut innehållet från tabellen *Avregistreradebilar.*

   Längst ner på sidan finns en knapp:
   - *Avsluta* ger användaren möjligheten att stänga ner sidan.

5.2.2 Skrotade bilar

Menyalternativet *Skrotade bilar* öppnar en ny sida i den *övre ramen* som nämndes i delavsnitt 5.1. Då den nya sidan är aktiv kan inte samma menyalternativ väljas igen förrän användaren stänger ner den aktuella sidan. Den nya sidan innehåller en utskrift från tabellen *Skrotade bilar*. Längst ner på sidan finns en knapp *Avsluta* som ger användaren möjligheten att stänga ner sidan.

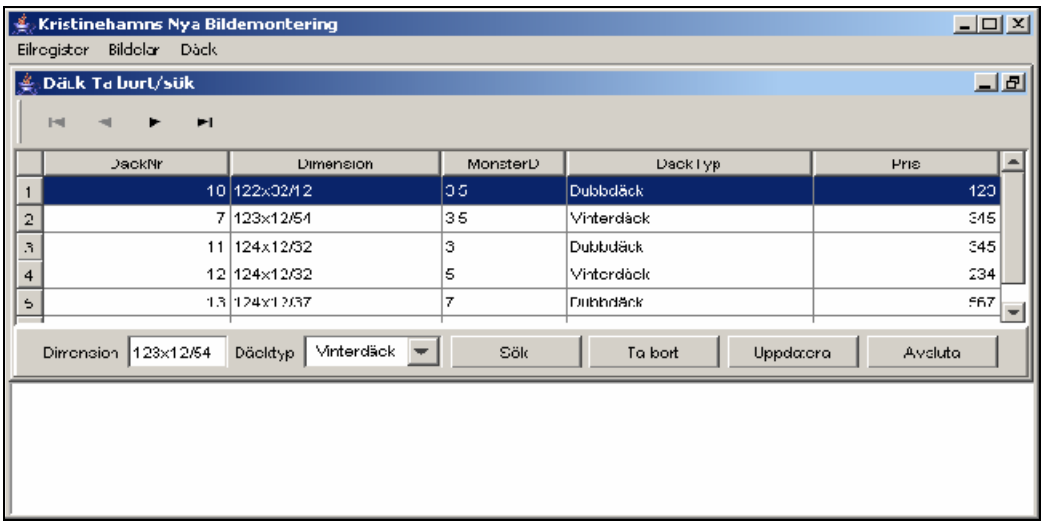
5.3 Däck

Menyn *Däck* innehåller en undermeny som består av två alternativ; *Lägg till* och *Ta bort/Sök* som ger användaren möjligheten att manipulera databasen *Däck*. Då användaren valt något av de ovanstående alternativen öppnas en ny sida i den *övre ramen* som nämndes i delavsnitt 5.1. Då den nya sidan är aktiv kan inte samma menyalternativ väljas igen förrän användaren stänger ner den aktuella sidan.

1. Alternativet *Lägg till*: Användaren har möjligheten att lägga till ett nytt däck i databasen *Däck*. Den nya sidan innehåller rullgardiner med fördefinierade värden och inmatningsrutor där användaren får mata in de värden som krävs för att lägga till ett nytt däck.

   Längst ner på sidan finns tre knappar:

   - *Spara* ger användaren möjligheten att spara de angivna värdena i databasen *Däck*.
   - *Töm fälten* ger användaren möjligheten att nollställa alla värden på sidan och uppdaterar utskriften.
   - *Avsluta* ger användaren möjligheten att stänga ner sidan.



Figur 5.3 Bild över Däck - Ta bort/Sök

2. Alternativet *Ta bort/Sök*: Användaren har möjligheten att söka eller ta bort ett däck från databasen *Däck*. Den nya sidan innehåller rullgardiner med fördefinierade värden och inmatningsrutor där användaren får mata in de värden som krävs för att söka efter ett däck i databasen. Då användaren fått sitt resultat på sökningen ges möjligheten att ta bort ett däck åt gången från databasen genom att markera den rad som ska tas bort och trycka på knappen *Ta bort* (se figur 5.3).

Längst ner på sidan finns fyra knappar:

- *Sök* ger användaren möjligheten att söka efter däck i databasen *Däck*.
- *Ta bort* ger användaren möjligheten att ta bort det markerade däcket i sökresultatet från databasen *Däck*.
- *Uppdatera* ger användaren möjligheten att nollställa alla värden på sidan och uppdaterar utskriften.
- *Avsluta* ger användaren möjligheten att stänga ner sidan.

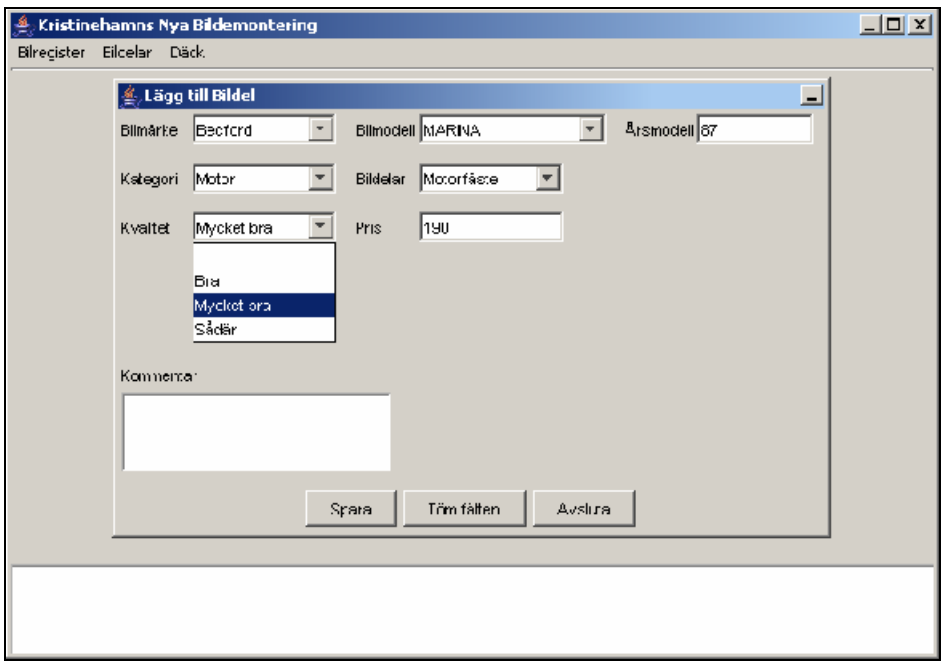
5.4 Bildelar

Menyn *Bildelar* innehåller en undermeny som består av två alternativ; *Lägg till*, *Ta bort/Sök* som ger användaren möjligheten att manipulera databasen *Bildelar*.
Då användaren valt något av de ovanstående alternativen öppnas en ny sida i den *övre ramen* som nämndes i delavsnitt 5.1. Då den nya sidan är aktiv kan inte samma menyalternativ väljas igen förrän användaren stänger ner den aktuella sidan.

1. Alternativet *Lägg till*: Användaren har möjligheten att lägga till en ny bildel i databasen *Bildel*. Den nya sidan innehåller rullgardiner med fördefinierade värden och inmatningsrutor där användaren får mata in de värden som krävs för att lägga till en ny bildel (se figur 5.4).

   Längst ner på sidan finns tre knappar:

   - *Spara* ger användaren möjligheten att spara de angivna värdena i databasen *Bildel*.
   - *Töm fälten* ger användaren möjligheten att nollställa alla värden på sidan och uppdaterar utskriften.
   - *Avsluta* ger användaren möjligheten att stänga ner sidan.



Figur 5.4 Bild över Bildel - Lägg till

2. Alternativet *Ta bort/Sök*: Användaren har möjligheten att söka eller ta bort en bildel från databasen *Bildel*. Den nya sidan innehåller rullgardiner med fördefinierade värden och inmatningsrutor där användaren får mata in de värden som krävs för att söka efter en bildel i databasen. Då användaren fått sitt resultat på sökningen ges möjligheten att ta bort en bildel åt gången från databasen genom att markera den rad som ska tas bort och trycka på knappen *Ta bort*.

   Längst ner på sidan finns fyra knappar:

- *Sök* ger användaren möjligheten att söka efter bildelar i databasen *Bildel*.
- *Ta bort* ger användaren möjligheten att ta bort den markerade bildelen i sökresultatet från databasen *Bildel*.
- *Uppdatera* ger användaren möjligheten att nollställa alla värden på sidan och uppdaterar utskriften.
- *Avsluta* ger användaren möjligheten att stänga ner sidan.

6 Teknisk beskrivning av databaserna

I detta kapitel kommer vi att beskriva vissa tekniska aspekter av våra databaser. I delavsnitt 6.1 tar vi upp anledningarna till varför vi har valt MySQL som vår databasserver och i delavsnitt 6.2 tar vi upp tänkandet bakom designen av vår databas.

6.1 Valet av MySQL Server

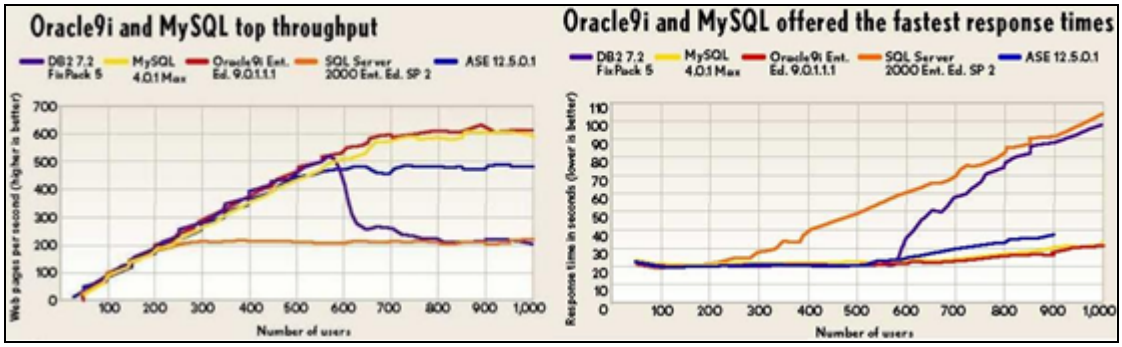
Det finns otaliga företag ute på marknaden som erbjuder olika typer av databaser. För att nämna några företag så har vi *IBM, Microsoft, MySQL, Oracle* och *Sybase*. Självfallet hävdar alla företag att deras databasprodukter är de bästa som marknaden har att erbjuda.

Att göra en egen jämförelse mellan MySQL och de ovannämnda företagens databasservrar är ett arbete för sig. Därför har vi baserat vårt val på tidigare undersökningar, bl.a. [3], som andra har gjort och vår egen erfarenhet av databasen *MySQL Server 4.1*, som har visat sig vara snabb, effektiv och enkel. Enkel i den bemärkelsen att inga komplicerade konfigurationer behövs göras för att komma igång med användandet av programmet.

En av de mest tilltalande aspekterna är att MySQL finns som gratisversion[2] vilken får användas privat. Däremot om MySQL-databasen ska användas kommersiellt måste en licens köpas av MySQL AB. Priset för *MySQL classic*, d.v.s versionen som inte stödjer funktionen InnoDB -stöd för bl.a. snabb och säker överföring av stora volymer data, användning av främmande nycklar, etc.- ligger på 2200 kr, och *MySQL pro* som stödjer InnoDB kostar 4400 kr.

I februari 2002 gjorde eWeek Magazine en stor prestandajämförelse [3] över de ledande databaservrerna *IBM DB2 7.2, Microsoft SQL Server 2000, MySQL-Max 4.0.1, Oracle 9i 9.0.1.1.1* och *Sybase ASE 12.5.0.1* och fann att Oracle och gratisversionen av MySQL var de två bästa bland de undersökta databaserna när det gällde genomströmning och svarstid (se figur 6.1).

[2] http://www.mysql.com



Figur 6.1 Genomströmningen och svarstiderna från eWeek undersökningen [3]

6.2 Utvecklingen av designen

Normalisering bygger på principen att bryta ner stora tabeller till flera mindre tabeller för att i slutändan få en bättre design. Detta uppnås genom att se till att tabellerna uppnår en viss normalform. För databaserna *Bilregister*, *Däck* och *Bildel* räckte det att uppnå den tredje normalformen för att förhindra transitivt beroende mellan attributerna. Det bör noteras att det även finns högre grader av normalformer. För att kunna uppfylla en högre normalform så måste man ha uppfyllt de underliggande normalformerna [4]. Här nedan följer en kort beskrivning av de normalformer som vi har använt oss av:

1NF: En relation där korsningen av varje rad och kolumn innehåller ett och endast ett värde.

2NF: Måste ha uppfyllt första normalformen och alla attribut som inte är en del av primärnyckeln måste vara fullständigt beroende av primärnyckeln.

3NF: Måste ha uppfyllt första och andra normalformen och ingen av de attribut som inte till tillhör primärnyckeln får vara transitivt beroende av primärnyckeln.

När vi har normaliserat våra tabeller måste vi implementera dessa i våra databaser *Bilregister*, *Bildel* och *Däck*. Under implementationen deklarerar vi våra tabeller och deras attribut, då dessa kan ta olika former.

Attributen kan vara av många olika datatyper, men vi har valt att begränsa oss till datatyperna char, varchar och int. Vi måste även deklarera vilka attribut som får och inte får ha null-värden och slutligen måste vi bestämma våra primär- och främmande nycklar.

Här nedan följer beskrivningen av de viktigaste punkterna ur våra databaser:

- ***Bilregister***

  Som tidigare har nämnts i delavsnitt 4.1 består databasen *Bilregister* av två tabeller; *Avregistrerade bilar* och *Skrotade bilar* som är identiska. Därför nöjer vi oss med att endast beskriva principen bakom tabellen *Avregistrerade bilar*.

  Attributet (se Bilaga A1) *RefNr* är av typen *int*, den är unik och alla bilar som avregistreras tilldelas ett sådant. Därför har vi valt att sätta det till primärnyckel. Notera att attributet *RegNr* också är unikt men vissa av de befintliga värdena som vi har fått av uppdragsgivaren är inte angivna och därför har vi varit tvungen att tillåta NULL-värden för attributet. Detta leder automatiskt till att RegNr blir oanvändbar som primärnyckel.

  Attributet *Bil* skall innehålla olika långa namn på de olika bilmärken och modeller som finns och måste därför vara dynamisk, därför har vi använt oss av typen *varchar*. *Varchar* har fördelar och nackdelar; en fördel är att den allokerar minne dynamiskt men har nackdelen att den är långsammare än den vanligare typen *char*.

- ***Däck***

  Däck innehåller endast tabellen *dack* (se bilaga B2) som har primärnyckeln *DackNr* av typen *int*. Det speciella med den är att vi använder oss av den inbyggda funktionen *AUTO_INCREMENT* som automatiskt tilldelar unika värden till vår primärnyckel. Dess fördel är att MySQL hanterar skapandet och borttagandet av dess värden, nackdelen är att den inte återanvänder samma värde trots att det har tagits bort vilket kan leda till stora värden i det långa loppet.

- ***Bildel***

  Bildel innehåller en stor mängd tabeller vilka har tagits fram genom normaliseringen av den ursprungliga tabellen som innehöll alla bildelar som existerar i en bilskrot. Som tidigare nämnts i delavsnitt 4.3 har vi grupperat tabellerna i större kategorier såsom motor och växellåda. Dessa kategorier är alla beroende av tabellen *bil* (se bilaga A3) då alla delar härrör just från en bil.

7 Teknisk beskrivning av användargränssnittet

I detta kapitel beskrivs de tekniska aspekterna av användargränssnittet. Kapitlet börjar med delavsnitt 7.1 som tar upp valet av språket Java och fortsätter med delavsnitt 7.2 som tar upp *Skalet* vilket följs av delavsnitt 7.3 och 7.4 som tar upp *Bilregister* och *Däck*. Delavsnitt 7.5 beskriver de tekniska aspekterna av *Bildel*.

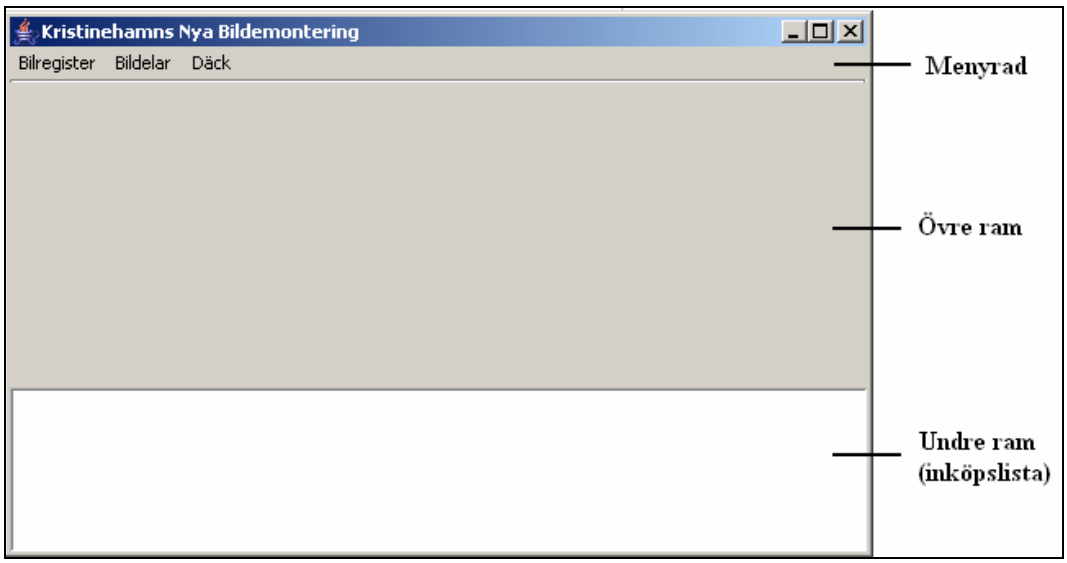
7.1 Val av språket Java

Fördelarna med språket Java är många men de viktigaste punkterna nämns här nedan:

- Språket är plattformsoberoende
- Språket är objektorienterat där allt är objekt, förutom grundläggande datatyper såsom tal och tecken.
- Språket erbjuder ett stort och väldokumenterat klassbibliotek [5].
- Språket erbjuder bra felhantering med tjänster såsom *garbage collector*.
- Dess syntax liknar syntaxen i C++ vilket underlättar för C++ programmerare att lära sig detta robusta språk.

Då vi inte hade programmerat i språket Java tidigare var den största fördelen att Java och C++ syntax liknar varandra och är lättare att sätta sig in i, då vi tidigare programmerat i C++. Som utvecklingsmiljö har vi valt Borland JBuilder 2005 Enterprise [6].

7.2 Skalet



Figur 7.1 Bild av programmets Skal

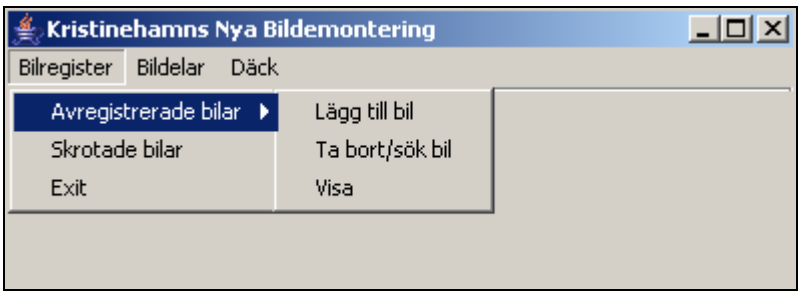
Som nämndes i delavsnitt 5.1 består *Skalet* av en yttre ram innehållande en menyrad, och de två inre ramarna *Övre ram* och *Undre ram* (se figur 7.1). Den yttre ramen är av klassen *JFrame* som initieras i början av koden. Med *JFrame* skapar vi ett fönster med en rubrik som har det klassiska Windowsutseendet, där användaren har möjligheten att minimera, maximera, ändra storlek och stänga ner fönstret och därmed programmet. Då vår yttre ram är skapad skapar vi en menyrad som kopplas samman med den yttre ramen, detta tas upp i delavsnitt 7.2.1.

I den yttre ramen skapar vi en arbetsyta, en s.k. *ContentPane*, som innehåller alla andra komponenter. För att kunna lägga in komponenter i *ContentPane* måste den först ha en ”strategi” för var de underliggande komponenterna skall placeras. Detta löses genom att använda *ContentPanes* inbyggda *LayoutManager* som innehåller färdigbyggda strategier för hur fönstren ska se ut, var komponenterna ska placeras och hur de ska bete sig. Det bör nämnas att varje arbetsyta har en inbyggd *LayoutManager* som ger programmeraren möjligheten att välja mellan de olika strategierna *FlowLayout*, *BorderLayout*, *GridLayout*, etc.

För vår *ContentPane* använder vi oss av strategin *BorderLayout* som ger oss möjligheten att fördela vår arbetsyta i fem delar, där komponenterna placeras ut längst den yttre ramens fyra kanter samt i den yttre ramens mitt. Denna strategi kompenserar automatiskt för eventuella förändringar i den yttre ramens storlek genom att den flyttar och eventuellt ändrar storlek proportionellt på de inre komponenternas placering så att GUI:t behåller sin ursprungliga design. Nu när vi har angett strategin för vår arbetsyta kan vi börja skapa vår *Övre ram* och vår *Undre ram* (se figur 7.1). Vi börjar med att skapa vår *Övre ram* genom att skapa en panel av klassen *JPanel* som vi placerar i den centrala delen av *ContentPane*. Som tidigare nämnts i delavsnitt 5.1 öppnas alla fönster som vi tar upp i kommande delavsnitt i denna ram, därför måste en strategi anges även här. För den här panelen valde vi *FlowLayout* som strategi, detta för att de enskilda komponenternas storlek inte ändras av den yttre ramens *LayoutManager* då den yttre ramen ändrar sin storlek. Nästa steg blir att skapa *Undre ram* vilket görs på samma sätt som *Övre ram* men placeras i skalets nedre del och kommer att innehålla *Inköpslistan*.

7.2.1 Menyn

För att skapa en meny krävs först en menyrad vilken skapas genom klassen *JMenuBar*. Då denna är skapad sätts den fast i den yttre ramen, och vi kan nu skapa våra menyer, submenyer och menyalternativ.



Figur 7.2 Bild över menyn

Vi börjar med att skapa menyerna *Bilregister*, *Bildelar* och *Däck* (se figur 7.2). För att skapa en meny används klassen *JMenu* som i sin tur kopplas till menyraden som redan är skapad. Nu när menyn är skapad kan vi placera våra menyalternativ i den. Menyalternativen skapas genom klassen *JMenuItem* vilken kan bestå av text eller bilder.

Ett menyalternativ kan i sin tur vara en submeny som innehåller egna menyalternativ. Ett sådant skapas också genom klassen *JMenu* och sätts fast i en befintlig meny. När användaren väljer ett sådant menyalternativ kommer submenyn att visas (se figur 7.2).

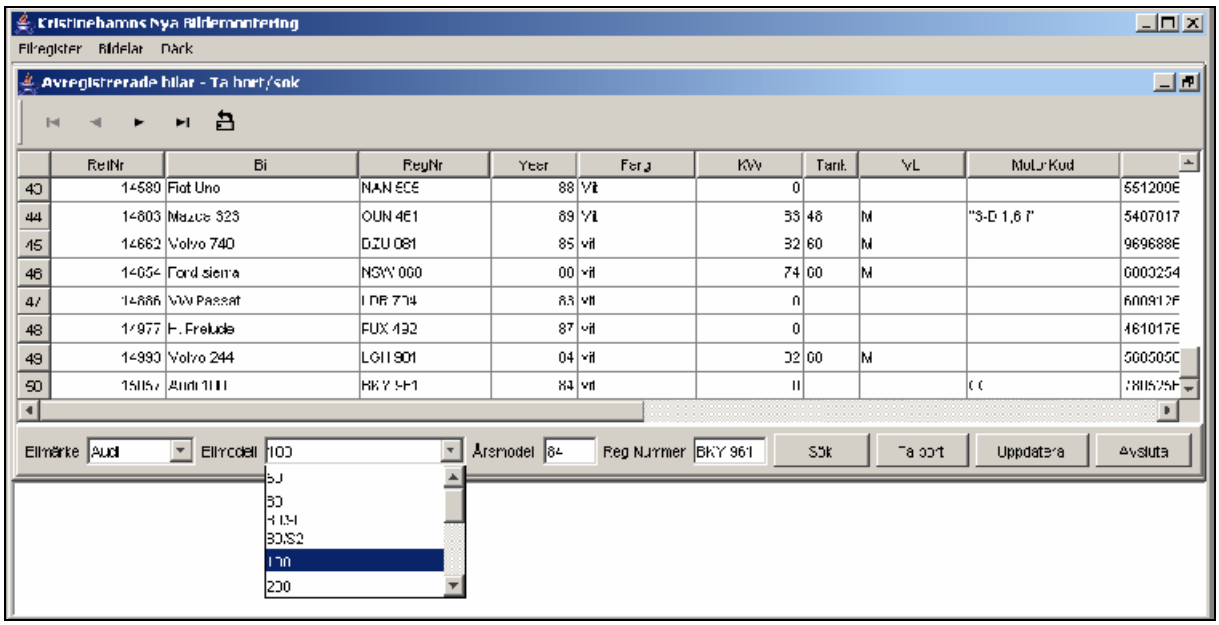
Det enda som återstår nu är att definiera en lyssnare och en händelsehanterare en s.k. *ActionListener* och en *ActionPerformed* till varje menyalternativ. *ActionListener* har till uppgift att ta emot händelser som uppstår när användaren väljer något av menyalternativen och skickar den vidare till *ActionPerformed*. Där aktiveras de lämpliga åtgärderna som t.ex. att öppna ett nytt fönster och avaktivera det valda menyalternativet för att hindra användaren från att välja samma menyalternativ mer än en gång åt gången.

7.3 Bilregister

I detta delavsnitt tas först de tekniska aspekterna för menyalternativet *Avregistrerade bilar* upp. Detta är indelat i *Ta bort/Sök, Lägg till* och *Visa* vilka tas upp i tur och ordning i delavsnitten 7.3.1-7.3.3. Därefter tas menyalternativet *Skrotade bilar* upp i delavsnitt 7.3.5.

7.3.1 Avregistrerade bilar – Ta bort/Sök

I delavsnitt 5.2.1 nämndes det att *Avregistrerade bilar* bestod av de tre alternativen *Lägg till, Ta bort/Sök* och *Visa* som var för sig öppnade en ny sida i *Övre ram*. För att göra detta möjligt måste den nya sidan skapas. Här kan vi dock inte använda oss av klassen *JFrame* som vi använde för att skapa skalet, då denna skulle öppna en ny sida utanför vårt skal. Därför använder vi oss av klassen *JInternalFrame* som skapar ett nytt fönster innanför *Övre ram* som finns i skalet. Då vi skapar en inre ram måste vi deklarera dess egenskaper såsom att kunna minimera, maximera, ändra storlek och stänga ner ramen vi skapar. Detta behövde vi inte göra med *JFrame* då den redan var fördefinierad. Som framgår av figur 7.3 har vi inte använt egenskapen stäng då vi vill att användaren ska använda sig av vår *Avsluta* knapp.



Figur 7.3 Bild av menyalternativet Avregistrerade bilar, funktionen Ta bort/Sök

- **Skapa arbetsyta**

  Nästa steg blir att skapa en arbetsyta som motsvarar skalets *ContentPane*. Detta görs med klassen *JPanel* där vi använder oss av strategin *BorderLayout* och placerar den i mitten av vår inre ram. För att det ska var möjligt måste även den inre ramen ges en strategi, vi har valt *BorderLayout*. Nu när inre ramens arbetsyta är skapad kan vi skapa de komponenter som den ska innehålla.

- **Skapa tabell**

  Eftersom tabeller har en tendens att bli stora och otympliga behöver vi använda oss av rullister som kan underlätta visualiseringen av utskriften. Till detta använder vi klassen *TableScrollPane* som i sig är en panel specialiserad för tabeller. Denna rullningspanel kopplar vi samman med den inre ramens arbetsyta. När detta är gjort skapar vi en tabell av den databasspecifika klassen *JdbTable* som i sin tur kopplas samman med rullningspanelen.

- **Skapa databas**

  För att skapa en databas måste man först anropa klassen *Database*. Denna skapar en virtuell databas som är länkad till *MySQL Server*. För att kunna använda den virtuella databasen skapar vi *queryDataSet* som hanterar den data som finns i databasen. När detta är gjort måste databasen kopplas till tabellen som skapats med *JdbTable*.

- **Skapa navigeringsmeny**

  Vi börjar med att skapa en panel som ska innehålla en navigeringmeny. Till detta använder vi klassen *JPanel* som ska vara belägen i arbetsytans övre del och ha strategin *GridLayout*. Denna strategi ger oss möjligheten att placera navigerinsmenyn i valfritt geografiskt område. Vi har valt att placera den i den vänstra sidan av vår arbetsyta.

  För att skapa navigeringsmenyn (se figur 7.3) använder vi klassen *JdbNavToolBar* som är inbyggd i utvecklingsmiljön *Borland JBuilder*. Denna klass har en mängd olika knappar inbyggda såsom *First*, *Prior*, *Next*, *Refresh* etc. vilka ger användaren möjligheten att navigera i tabellen.

- **Skapa sökfält**

  Nästa steg blir att skapa en ny panel där användaren kan hämta data. Som tidigare nämnts skapar vi först en panel av klassen *JPanel*, placerar den i inre ramens nedre del (se figur 7.3) och sätter *FlowLayout* som dess strategi. Nu kan vi skapa de komponenter som krävs för att ge användaren möjligheten att söka bilar, ta bort bilar och stänga ner den inre ramen. För *Bilmärke* och *Bilmodell* använder vi oss av rullgardiner vilka skapas med klassen *JComboBox*. Eftersom *Bilmärke* och *Bilmodell* är sammansatt till en kolumn i vår databas, måste vi sammanfoga de valda värdena från rullgardinerna till ett. Förutom rullgardinerna skapar vi även de två textfälten där användaren ska mata in årsmodell och registreringsnummer på den sökta bilen, detta görs genom att använda klassen *TexFiled*. För att göra GUI:t så användarvänligt som möjligt har vi även lagt till etiketter (av klassen *JLable*) bredvid rullgardinerna och textfälten för att informera användaren vad dessa ska användas till.

- **Knappen Sök**

  Nästa steg blir att söka efter data man är intresserad av ur databasen. Detta görs genom att hämta informationen användaren matat in i sökfälten. Dessa inhämtade värden sparas som variabler, vilka inbäddas i en SQL-fråga och skickas till MySQL Server, då användare trycker på *Sök* knappen (se figur 7.3). Denne i sin tur uppdaterar den virtuella databasen som uppdaterar den virtuella tabellen.

  Det är viktigt att komma ihåg att när en SQL-fråga skickas måste tidigare uppkopplingar stängas ner, detta för att den virtuella tabellen ska kunna uppdateras med det nya sökresultatet.

- **Knappen Ta bort**

  Nästa steg blir att ta bort data ur databasen, detta görs genom att markera den rad användaren vill ta bort och klicka på knappen Ta bort. Då detta sker aktiveras den virtuella databasen *Skrotadebilar* och en ny rad skapas. Nu kopieras den markerade raden ur *Avregistreradebilar*, läggs in i den nyss skapade raden och sparas i MySQL Server. Då ändringarna i *Skrotadebilar* sparats tas den markerade raden i *Avregistreradebilar* bort och ändringen sparas i MySQL Server.

- **Knappen Uppdatera**

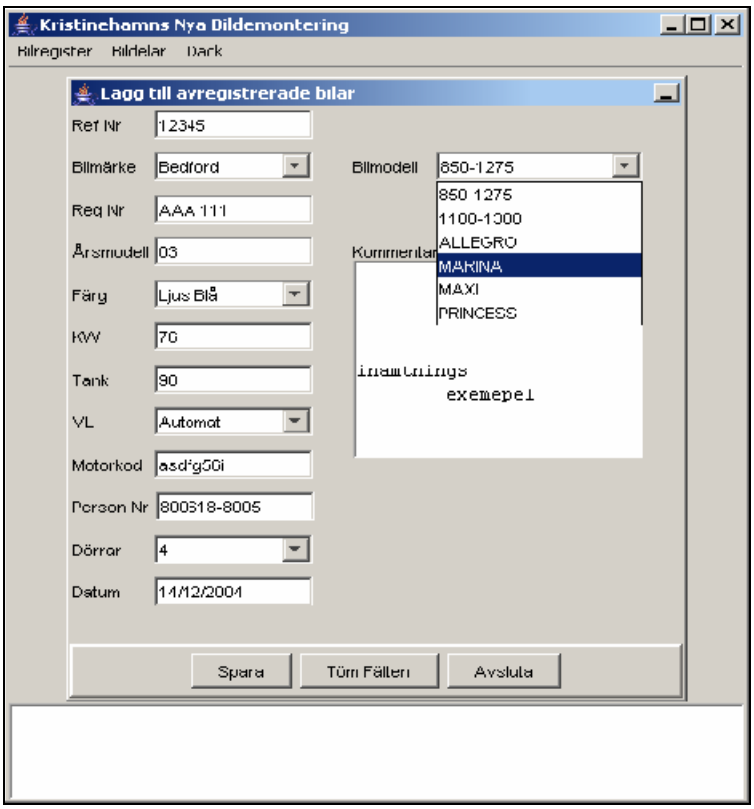
  Uppdateringen av databasen *Avregistreradebilar* sker då användaren trycker på *Uppdatera*-knappen. Då detta görs stängs den aktiva uppkopplingen ner och en ny SQL-fråga skickas till MySQL Server för att uppdatera den aktuella sidan. Därefter nollställs även sökfälten. Nu kan användaren se de uppdaterade värdena och göra en ny sökning.

- **Knappen Avsluta**

  Då användaren trycker på knappen avsluta, kommer den aktuella sidan att stängas ner och menyalternativet *Avregistrerade bilar - Ta bort/sök* aktiveras och kan nu åter väljas.

7.3.2 Avregistrerade bilar – Lägg till

Som nämndes i delavsnitt 7.3.1 skapar vi en inre ram som öppnas i skalets *Övre ram*.



Figur 7.4 Bild på Avregistrerade bilar- Lägg till

- **Skapa arbetsyta**

  Nästa steg blir att skapa en arbetsyta med strategin BorderLayout som motsvarar skalets *ContentPane*. I denna arbetsyta placeras två paneler över varandra. Den övre panelen placeras i mitten av vår arbetsyta och har *xYLayout* som strategi för att möjliggöra placering av komponenter vid valfria koordinater.

  Nu kan vi skapa de komponenter som krävs för att ge användaren möjligheten att mata in bilar i databasen *Avregistreradebilar*. För *Bilmärke*, *Bilmodell*, *Färg*, *VL* och *Dörrar* använder vi oss av rullgardiner. Förutom rullgardinerna skapar vi även de åtta textfälten *Ref Nr*, *Reg Nr*, *Årsmodell*, *KW*, *Tank*, *MotorKod*, *PNr* och *Datum*. Vi använder även en textarea *Kommentar* där användaren kan mata in eventuella kommentarer.

Den undre panelen placeras i nedre delen av vår arbetsyta och har *FlowLayout* som strategi och innehåller knapparna *Spara*, *Töm fält* och *Avsluta*.

- **Knappen Spara**

  Nästa steg blir att mata in data i databasen *Avregistreradebilar*, detta görs genom att hämta informationen användaren matat in i inmatningsfälten. Dessa inhämtade värden sparas som variabler vilka inbäddas i en SQL-fråga och skickas till MySQL Server då användaren trycker på *Spara* knappen (se figur 7.4). Om inmatningen lyckas får användaren ett meddelande om detta, i annat fall ges ett felmeddelande.

- **Knappen Töm fält**

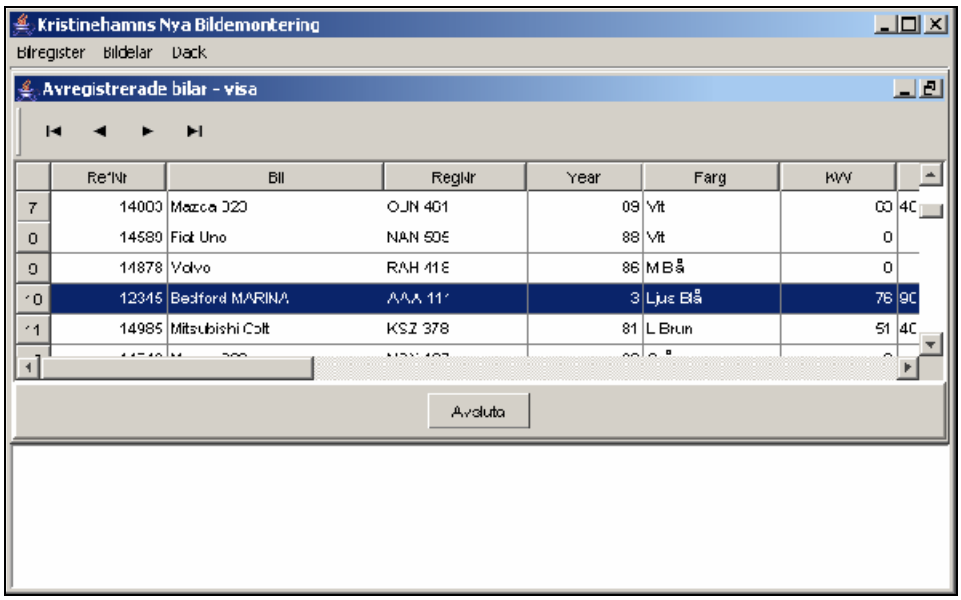
  Knappen *Töm fält* har till uppgift att nollställa inmatningsfälten.

- **Knappen Avsluta**

  Då användaren trycker på knappen avsluta, kommer den aktuella sidan att stängas ner och menyalternativet Avregistrerade bilar - *Lägg till* aktiveras och kan nu åter väljas.

7.3.3 Avregistrerade bilar – Visa

Som nämndes i delavsnitt 7.3.1 skapar vi en inre ram som öppnas i skalets *Övre ram*.



Figur 7.5 Bild på Avregistrerade bilar – Visa

- **Skapa arbetsyta**

  Nästa steg blir att skapa en arbetsyta med strategin BorderLayout som motsvarar skalets *ContentPane*. I denna arbetsyta placeras två paneler och en tabell. Den första panelen placeras i övre delen av vår arbetsyta och har *GridLayout* som strategi och innehåller en navigeringsmeny av klassen *JdbNavToolBar*. Denna klass har en mängd olika knappar inbyggda såsom *First*, *Prior*, *Next*, *Refresh* etc. som ger användaren möjligheten att navigera i tabellen.

  Den andra panelen placeras i nedre delen av vår arbetsyta och har *FlowLayout* som strategi och innehåller knappen *Avsluta*.

- **Skapa tabell**

  Nästa steg blir att skapa tabellen som nämndes i ovanstående punkt. För detta används en rullningspanel av klassen *TableScrollPane*. Denna rullningspanel kopplar vi samman med arbetsytan. När detta är gjort skapar vi en tabell av klassen *JdbTable* som i sin tur kopplas samman med rullningspanelen.

- **Skapa databas**

  För att skapa en databas måste man först anropa klassen *Database*. Denna skapar en virtuell databas som är länkad till *MySQL Server*. För att kunna använda den virtuella databasen skapar vi *queryDataSet* som hanterar den data som finns i databasen. När detta är gjort måste databasen kopplas till tabellen som skapats med *JdbTable*.

  Varje gång Menyalternativet *Avregistreradebilar – Visa* väljs skickas en SQL-fråga till MySQL Server för att hämta de senaste värdena ur tabellen *Avregistreradebilar*, och skrivs ut i tabellen som databasen är kopplad till.

- **Knappen Avsluta**

  Då användaren trycker på knappen avsluta, kommer den aktuella sidan att stängas ner och menyalternativet *Avregistrerade bilar - Visa* aktiveras och kan nu åter väljas (se figur 7.5).

7.3.4 Skrotade bilar

Skapas precis som *Avregistrerade bilar – Visa* i delavsnitt 7.3.3 , och därför har vi valt att inte beskriva den en gång till (se figur 7.5).

7.4 Däck

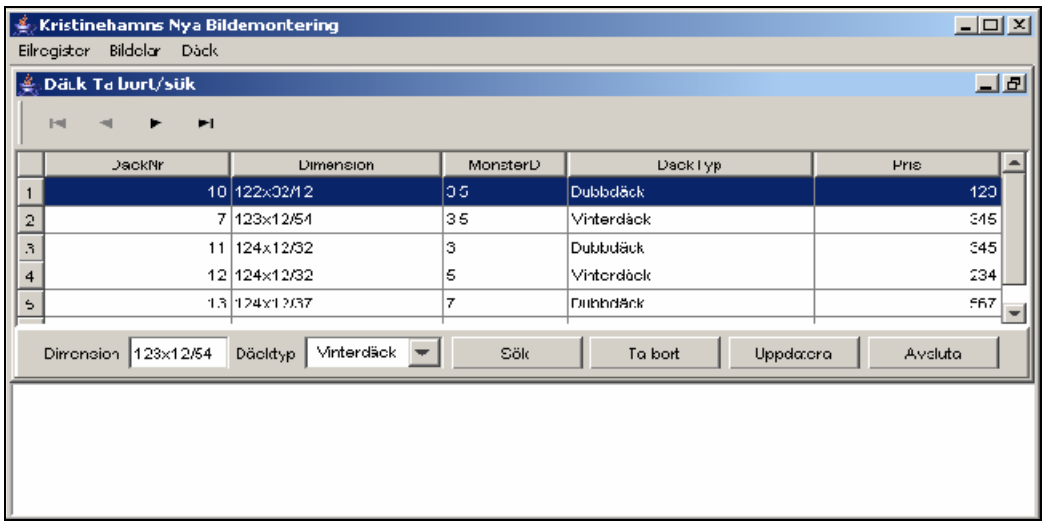
I detta delavsnitt tas de tekniska aspekterna upp för menyalternativen *Däck* som är indelad i *Ta bort/sök* och *Lägg till* vilka tas upp i delavsnitten 7.4.1 och 7.4.2.

7.4.1 Däck – Ta bort/Sök

Som nämndes i delavsnitt 7.3.1 skapar vi en inre ram som öppnas i skalets *Övre ram*. Nästa steg blir att skapa en arbetsyta med strategin *BorderLayout* som motsvarar skalets *ContentPane*. I denna arbetsyta placeras två paneler och en tabell. Den första panelen placeras i övre delen av vår arbetsyta och har *GridLayout* som strategi och innehåller en navigerings meny av klassen *JdbNavToolBar*.

Den andra panelen placeras i nedre delen av vår arbetsyta och har *FlowLayout* som strategi och innehåller sökfältet som består av knapparna *Sök*, *Ta bort*, *Uppdatera* och *Avsluta*. Panelen innehåller även en rullgardin där användaren kan välja däck typ och ett textfält för inmatning av däckens dimension.

Databasen, tabellen och knapparna *Sök*, *Ta bort*, *Uppdatera* och *Avsluta* skapas på samma sätt som i delavsnitt 7.3.1 (se figur 7.6).



Figur 7.6 Bild på Däck – Ta bort/Sök

7.4.2 Däck – Lägg till

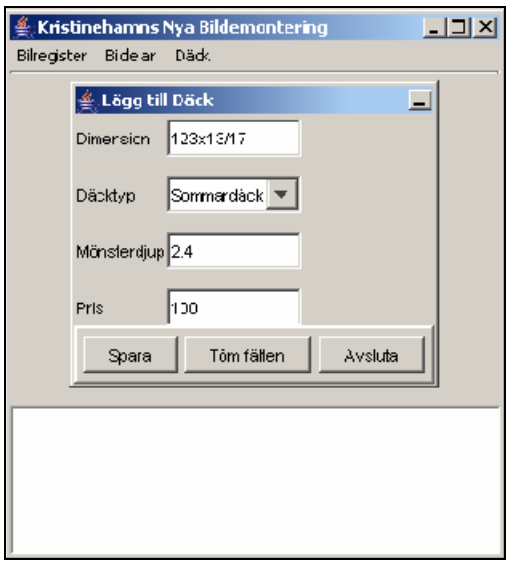
Som nämndes i delavsnitt 7.3.1 skapar vi en inre ram som öppnas i skalets *Övre ram*.

- **Skapa arbetsyta**

  Nästa steg blir att skapa en arbetsyta med strategin BorderLayout som motsvarar skalets *ContentPane*. I denna arbetsyta placeras två paneler över varandra. Den övre panelen placeras i mitten av vår arbetsyta och har *xYLayout* som strategi.

  Nu kan vi skapa de komponenter som krävs för att ge användaren möjligheten att mata in däck i databasen *Däck*.

  För *Däck typ* använder vi oss av en rullgardin. Förutom rullgardinen skapar vi även de tre textfälten *Dimension*, *Pris* och *Mönsterdjup*. Den undre panelen placeras i nedre delen av vår arbetsyta och har *FlowLayout* som strategi och innehåller knapparna *Spara*, *Töm fält* och *Avsluta*. Dessa skapas på samma sätt som i delavsnitt 7.3.2 (se figur 7.7).



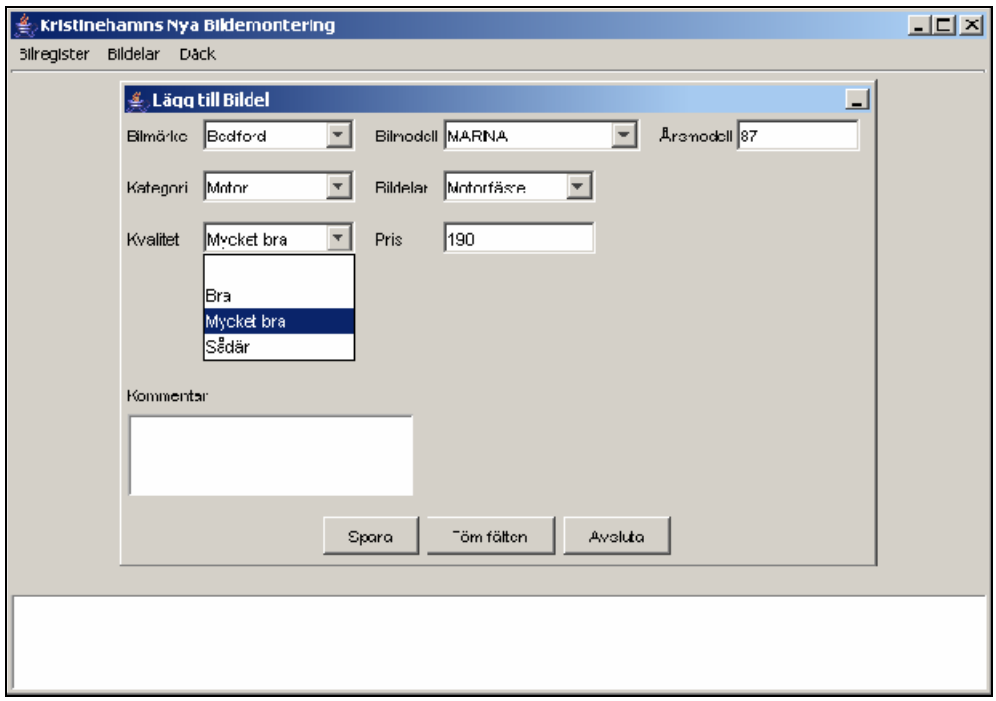
Figur 7.7 Bild på Däck – Lägg till

7.5 Bildelar

I detta delavsnitt tas de tekniska aspekterna upp för menyalternativen *Bildelar* som är indelad i *Ta bort/sök* och *Lägg till* vilka tas upp i delavsnitten 7.5.1 och 7.5.2.

7.5.1 Bildelar – Lägg till

Som nämndes i delavsnitt 7.3.1 skapar vi en inre ram som öppnas i skalets *Övre ram*.



Figur 7.8 Bild på Bildel - Lägg till

- **Skapa arbetsyta**

  Nästa steg blir att skapa en arbetsyta med strategin BorderLayout som motsvarar skalets *ContentPane*. I denna arbetsyta placeras två paneler över varandra. Den övre panelen placeras i mitten av vår arbetsyta och har *xYLayout* som strategi.

  Nu kan vi skapa de komponenter som krävs för att ge användaren möjligheten att mata in bildelar i databasen *Bildel*.

  För *Bilmärke, Bilmodell, Kategori, Bildelar, VL-Typ* och *Kvalitet* använder vi oss av rullgardiner. Förutom rullgardinerna skapar vi även de två textfälten *Pris* och *Årsmodell*.

Den undre panelen placeras i nedre delen av vår arbetsyta och har *FlowLayout* som strategi och innehåller knapparna *Spara*, *Töm fält* och *Avsluta*.

- **Knappen Spara**

  Nästa steg blir att mata in data i databasen *Bildel*, detta görs genom att hämta informationen användaren matat in i inmatningsfälten. Dessa inhämtade värden sparas som variabler vilka inbäddas i en SQL-fråga och skickas till MySQL Server då användaren trycker på *Spara* knappen (se figur 7.8). Primär- och främmandenycklarna för alla de olika tabellerna i databasen *Bildel* tas fram och anges med hjälp av en slumpgenerator som utnyttjar datum och tid för sitt startvärde för att försäkra sig om att dessa värden blir unika. Om inmatningen lyckas får användaren ett meddelande om detta, i annat fall ges ett felmeddelande.

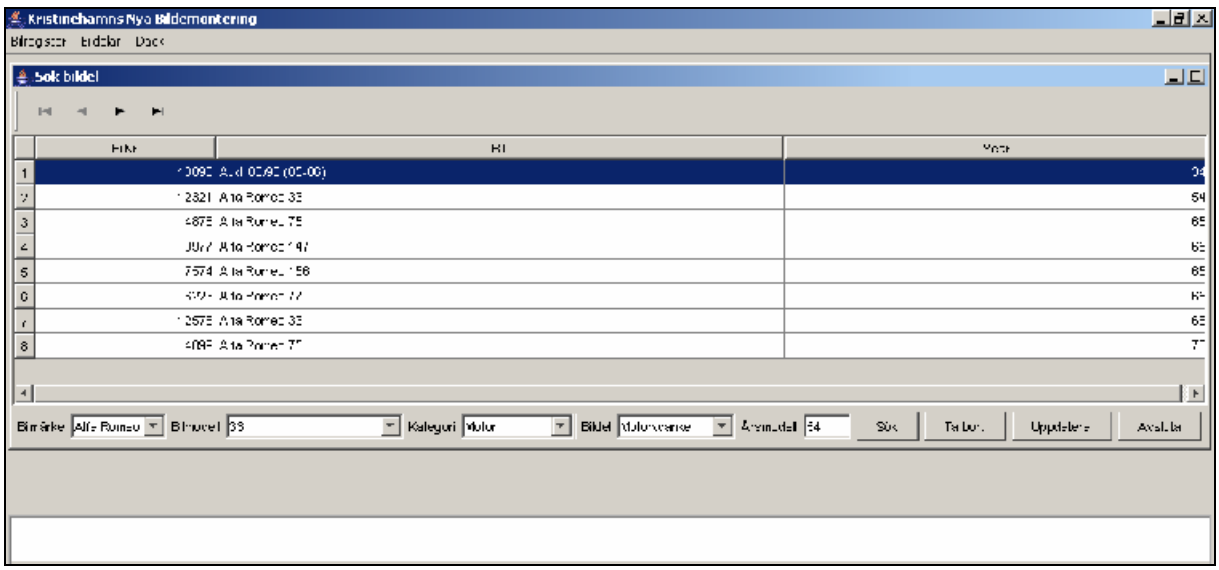
  Knapparna *Töm fälten* och *Avsluta* skapas på samma sätt som i delavsnitt 7.3.2 (se figur 7.8).

7.5.2 Bildelar – Ta bort/Sök

Som nämndes i delavsnitt 7.3.1 skapar vi en inre ram som öppnas i skalets *Övre ram*. Nu när den inre ramen är skapad blir nästa steg att skapa en arbetsyta med strategin *BorderLayout* som motsvarar skalets *ContentPane*. I denna arbetsyta placeras två paneler och en tabell. Den första panelen placeras i övre delen av vår arbetsyta och har *GridLayout* som strategi och innehåller en navigerings meny av klassen *JdbNavToolBar*.

Den andra panelen placeras i nedre delen av vår arbetsyta och har *FlowLayout* som strategi och innehåller sökfältet som består av knapparna *Sök*, *Ta bort*, *Uppdatera* och *Avsluta*. Panelen innehåller även de fyra rullgardinerna *Bilmärke*, *Bilmodell*, *Kategori* och *Bildelar* där användaren kan välja bil och bildel. Det finns även ett textfält för inmatning av bilens årsmodell.

Databasen, tabellen och knapparna *Sök*, *Ta bort*, *Uppdatera* och *Avsluta* skapas på samma sätt som i delavsnitt 7.3.1 (se figur 7.9).



Figur 7.9 Bild på Bildel – Ta bort/Sök

8 Kopplingen mellan MySQL och Java

För att skapa en koppling mellan vårt Javabaserade användargränssnitt och vår MySQL Server krävs det en *Java DataBase Connector* (JDBC) drivrutin. Dessa drivrutiner är programberoende och därför måste rätt drivrutin installeras. I vårt fall använder vi oss av programmeringsmiljön *Borland JBuilder 2005 Enterprise* för att implementera GUI:t och databasservern *MySQL Server 4.1* för operativsystemet *MS Windows*. Den rekommenderade drivrutinen för *JDBC* heter *MySQL Connector/J* vilken vi använder oss av.

I delavsnitt 8.1 beskrivs installation av drivrutinen och i delavsnitt 8.2 ges anvisningar för testning av installationen.

8.1 Installation av drivrutin

Här nedan följer en beskrivning av drivrutinens installation för Windows XP:

1. Installera programmeringsmiljön och databasen.

2. Installera drivrutinen *mysql-connector-java-3.0.16-ga-bin.jar*[3] som är den senast stabila versionen av *MySQL Connector/J*-serien.

3. Placera drivrutinen i *mysql* katalogen.

4. Placera även en kopia av drivrutinen i katalogen *JBuilder2005\lib\ext*. Denna katalog innehåller alla externa bibliotek som initieras vid kompilering och därmed möjliggörs en koppling mellan *JBuilder* och *MySQL*.

5. Initiera miljövariablerna genom att klicka på
   Start → Kontrollpanelen → System → Avancerat → Miljövariabler
   Systemvariabler: *ny*
   Variabelnamn: *classpath*
   Variabelvärde: *.;c:\mysql\mysql-connector-java.jar;*

[3] http://dev.mysql.com/downloads/connector/j/3.0.html

8.2 Testning av installation

För att testa installationen som beskrev i delavsnitt 8.1 följ nedanstående punkter:

I. För att bekräfta uppdateringen av miljövariablerna starta *Kommandotolken* i Windows och skriv kommandot: *set*

II. Nu ska uppkopplingen testas, detta görs genom att kompilera koden (se figur 8.1) i *Kommandotolken* genom att skriva följande rad i katalogen där filen befinner sig:
javac <filnamn>.java

III. Nu när vi har kompilerat koden ska vi exekvera den genom att skriva:
Java <filnamn>

IV. Om allt gått bra fås utskriften:
=> *LOADING DRIVER:*
=> *OK*
=> CONNECTION*:*
=> *OK*

```
//Initiated libraries
import java.sql.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class TestMysql
{
  public static void main(String args[]) {
    try {
      /* Test loading driver */
      String driver = "com.mysql.jdbc.Driver";

      System.out.println( "\n=> loading driver:" );
      Class.forName( driver ).newInstance();
      System.out.println( "OK" );

      /* Test the connection */
      String url = "jdbc:mysql://localhost/test";

      System.out.println( "\n=> connecting:" );
      DriverManager.getConnection( url, "", "" );
      System.out.println( "OK" );
    }
    catch( Exception x ) {
      x.printStackTrace();
    }
  }
}
```

Figur 8.1 Kod för testning av uppkoppling till MySQL Server

9 Slutsatser

Det har varit en utmaning att utveckla det här programmet, då vi i stort sätt hade fria händer att utveckla designen, och eftersom vi inte tidigare arbetat med att utveckla ett program av denna storleksordning.

Ett av de svåraste momenten var att implementera gränssnittet då vi inte tidigare programmerat i språket Java. Detta ledde till att en hel del tid gick åt att lära sig och hitta i Javas klassbibliotek, som för övrigt är ett mycket bra och väldokumenterad hjälpmedel.

Under implementeringens gång har vi stött på många oförväntade problem såsom skapandet av unika nycklar för våra tabeller i databasen *Bildel* och uppdateringen av våra tabeller. Detta tog tid att överkomma, vilket i sin tur ledde till att implementeringen tog längre tid än beräknat och funktionen *Inköpslista* ströks ur prototypen. Då vi har planer på att vidareutveckla programmet kommer denna funktion att implementeras vid senare tillfälle. Andra funktioner som skulle kunna implementeras är Telefonbok, ett register över alla bildelar som har sålts och tidpunkten för dess försäljning, en funktion som skriver ut kvitton till en skrivare. Endast fantasin begränsar programmets utveckling.

De färdiga momenten i programmet har genomgått en extensiv testning av både oss och arbetsgivaren som har resulterat i att alla de krav som framställs i kapitel 3 har uppfyllts.

Referenser

[1] http://www.sbrservice.se , 17 Dec. 04

[2] http://www.mysql.com , 17 Dec. 04

[3] http://www.eweek.com/article2/0,4149,293,00.asp, 17 Dec. 04

[4] http://www.cs.kau.se/cs/education/courses/davb04/lectures/normalisering.ppt, 17 Dec. 04

[5] http://java.sun.com/j2se/1.4.2/docs/api/index.html 17 Dec. 04

[6] http://www.borland.com/jbuilder 17 Dec. 04

Bibliografi

I. Thomas Counolly, Carolyn Begg, Anne Strachan, *Database System,* A Practical Approach to Design, Implementation, and Management Second Edition , Addison-Wesley, 1999.

II. Ramez Elmasri, Shamkant B. Navathe, Fundementals of Database Systems Forth Edition, Addison-Wesley, 2003.

III. Patrick O'Neil, Elizabeth O'Neil, *DB Principles, Programing, and Performance Second Edition*, Morgan Kaufmann Publishing, 2001.

IV. Greg Riccardi, *Principles of Database Systems with Internet and Java Application*, Addison-Wesley, 2001.

V. Laura Lemay, Rogers Cadenhead, *Lär dig Java 2 på 3 veckor*, Sams Publishing, 1999.

VI. Jan Skansholm, *Java direkt med Swing Fjärde Upplaga*, Studentlitteratur, 2003.

VII. H. M. Deitel, P. J. Deitel, *Java How to Program Fifth Edition*, Prentice Hall, 2003.

VIII. Ashton Hobbs, *Teach Yourself Database Programing with JDBC in 21 days*, Sams.net Publishing, 1997

A Relationsdatamodell

A.1 Mappning av E/R-modellen till relationsdatamodellen bilregister och dess beskrivning.

Avregistrerade bilar och *skrotade bilar* är två identiska entiteter som mappas till två identiska tabeller. Därför listas deras attribut endast en gång.

RefNr	Beskrivning:	Unikt nummer som fås av bilvision
	Datatyp:	Int
	Obligatorisk:	Ja
Bil	Beskrivning:	Bilens märke och modell
	Datatyp:	Varchar(30)
	Obligatorisk:	Ja
RegNr	Beskrivning:	Bilens registreringsnummer
	Datatyp:	Char(8)
	Obligatorisk:	Ja
Year	Beskrivning:	Bilens årsmodell
	Datatyp:	int
	Obligatorisk:	ja
Datum	Beskrivning:	Datumet då skroten tagit emot bilen
	Datatyp:	Varchar(12)
	Obligatorisk:	Nej (vissa av de tidigaste bil, saknar Datum)
Farg	Beskrivning:	Bilens färg
	Datatyp:	Varchar(15)
	Obligatorisk:	Ja

KW	Beskrivning:	Bilens motoreffekt
	Datatyp:	Int
	Obligatorisk:	Nej
Dorr	Beskrivning:	Antal dörrar på bilen
	Datatyp:	Int
	Obligatorisk:	Ja
Tank	Beskrivning:	Tankens storlek
	Datatyp:	Varchar(15)
	Obligatorisk:	Nej
VL	Beskrivning:	Typ av växellåda i bilen
	Datatyp:	Varchar(15)
	Obligatorisk:	Ja
MotorKod	Beskrivning:	Bilens motor kod
	Datatyp:	Varchar(20)
	Obligatorisk:	Nej
PNr	Beskrivning:	Bilägarens personnummer
	Datatyp:	Int
	Obligatorisk:	Nej
Komment	Beskrivning:	Skrotens kommentarer om den inlämnade bilen
	Datatyp:	Varchar(150)
	Obligatorisk:	Nej
Nycklar		
Primär:	RefNr	
Kandidat:	RefNr	
Främmande:	-	

A.2 Mappning av E/R-modellen till relationsdatamodellen däck och dess beskrivning

Däck har endast en entitet och mappas till en tabell, här listas dess attribut.

DackNr	Beskrivning:	Unikt nummer för däck
	Datattyp:	Int
	Obligatorisk:	Ja
Dimension	Beskrivning:	Däckens dimension
	Datattyp:	Char(12)
	Obligatorisk:	Ja
MonsterD	Beskrivning:	Däckens mönsterdjup i mm
	Datattyp:	Float
	Obligatorisk:	Nej
DackTyp	Beskrivning:	Anger typen av däck -sommar-, vinter- eller dubbdäck.
	Datattyp:	/*Int selectedindex(0,1,2)*/ varchar(12)
	Obligatorisk:	Ja
Pris	Beskrivning:	Priset på däck
	Datattyp:	Int
	Obligatorisk:	Ja
Nycklar		
Primär:	DackNr	
Kandidat:	DackNr	
Främmande:	-	

A.3 Mappning av E/R-modellen Bildelar till relationsdatamodellen Bildelar

Entiteter	Attribut
Bil	Namn, Modell, Year, BilNr
Motor	BilNr, MotorNr
MotorFaste	MFasteNr, MotorNr, Pris, Kvalitet, Komment
MotorWankel	MWankelNr, MotorNr, Pris, Kvalitet, Komment
VaxelLada	BilNr, VLNr
VLBalk	VLBalkNr, VLNr, Pris, Kvalitet, Komment
VLFaste	VLFasteNr, VLNr, Pris, Kvalitet, Komment

A.4 Beskrivning av de olika relationerna i DB bildelar.

Entiteten *Bil* mappas till en tabell, här listas dess attribut.

Bil	Beskrivning:	Bilen märke och modell
	Datatyp:	Varchar(30)
	Obligatorisk:	Ja
Year	Beskrivning:	Bilens årsmodell
	Datatyp:	Int
	Obligatorisk:	Ja
BilNr	Beskrivning:	Unikt nummer som tilldelas varje bil
	Datatyp:	Int
	Obligatorisk:	ja
Nycklar:		
Primär:	BilNr	
Kandidat:	BilNr	
Främmande:	-	

Entiteten *Motor* är även en relation mellan entiteten *Bil* och alla entiteter tillhörande kategorin motor**,** som mappas till en tabell. Här listas dess attribut.

BilNr	Beskrivning:	Unikt nummer som tilldelas varje bil
	Datatyp:	Int
	Obligatorisk:	Ja
MotorNr	Beskrivning:	Unikt nummer som tilldelas varje motor
	Datatyp:	Int
	Obligatorisk:	Ja
Nycklar:		
Primär:	(BilNr, MotorNr)	
Kandidat:	(BilNr, MotorNr)	
Främmande:	BilNr, MotorNr	

Entiteten *MotorFaste* mappas till en tabell, här listas dess attribut.

MFasteNr	Beskrivning:	Unikt nummer som tilldelas varje motorfäste
	Datatyp:	Int
	Obligatorisk:	Ja
MotorNr	Beskrivning:	Unikt nummer som tilldelas varje motor
	Datatyp:	Int
	Obligatorisk:	Ja
Pris	Beskrivning:	Priset på entiteten
	Datatyp:	Int
	Obligatorisk:	Ja
Kvalitet	Beskrivning:	Entitetens kvalitet
	Datatyp:	Varchar(15)
	Obligatorisk:	Ja
Komment	Beskrivning:	Användarens kommentar
	Datatyp:	Varchar(150)
	Obligatorisk:	Nej
Nycklar:		
Primär:	MFasteNR	
Kandidat:	MFasteNR	
Främmande:	MotorNr	

Entiteten *MotorWankel* mappas till en tabell, här listas dess attribut.

MWankelNr	Beskrivning:	Unikt nummer som tilldelas varje motorwankel
	Datatyp:	Int
	Obligatorisk:	Ja
MotorNr	Beskrivning:	Unikt nummer som tilldelas varje motor
	Datatyp:	Int
	Obligatorisk:	Ja
Pris	Beskrivning:	Priset på entiteten
	Datatyp:	Int
	Obligatorisk:	Ja
Kvalitet	Beskrivning:	Entitetens kvalitet
	Datatyp:	Varchar(15)
	Obligatorisk:	Ja
Komment	Beskrivning:	Användarens kommentar
	Datatyp:	Varchar(150)
	Obligatorisk:	Nej
Nycklar:		
Primär:	MWankelNR	
Kandidat:	MWankelNR	
Främmande:	MotorNr	

Entiteten *Växellåda* är även en relation mellan entiteten *Bil* och alla entiteter tillhörande kategorin växellåda**,** som mappas till en tabell. Här listas dess attribut.

BilNr	Beskrivning:	Unikt nummer som tilldelas varje bil
	Datatyp:	Int
	Obligatorisk:	Ja
VLNr	Beskrivning:	Unikt nummer som tilldelas varje växellåda
	Datatyp:	Int
	Obligatorisk:	Ja
Nycklar:		
Primär:	(BilNr, VLNr)	
Kandidat:	(BilNr, VLNr)	
Främmande:	BilNr, VLNr	

Entiteten *VLFaste* mappas till en tabell, här listas dess attribut.

VLFasteNr	Beskrivning:	Unikt nummer som tilldelas varje växellåda fäste
	Datatyp:	Int
	Obligatorisk:	Ja
VLNR	Beskrivning:	Unikt nummer som tilldelas varje växellåda
	Datatyp:	Int
	Obligatorisk:	Ja
Pris	Beskrivning:	Priset på entiteten
	Datatyp:	Int
	Obligatorisk:	Ja
Kvalitet	Beskrivning:	Entitetens kvalitet
	Datatyp:	Varchar(15)
	Obligatorisk:	Ja
Komment	Beskrivning:	Användarens kommentar
	Datatyp:	Varchar(150)
	Obligatorisk:	Nej
Nycklar:		
Primär:	VLFasteNr	
Kandidat:	VLFasteNr	
Främmande:	VLNr	

Entiteten *VLBalk* mappas till en tabell, här listas dess attribut.

VLBalkNr	Beskrivning:	Unikt nummer som tilldelas varje växellåda balk
	Datatyp:	Int
	Obligatorisk:	Ja
VLNR	Beskrivning:	Unikt nummer som tilldelas varje växellåda
	Datatyp:	Int
	Obligatorisk:	Ja
Pris	Beskrivning:	Priset på entiteten
	Datatyp:	Int
	Obligatorisk:	Ja
Kvalitet	Beskrivning:	Entitetens kvalitet
	Datatyp:	Varchar(15)
	Obligatorisk:	Ja
Komment	Beskrivning:	Användarens kommentar
	Datatyp:	Varchar(150)
	Obligatorisk:	Nej
Nycklar:		
Primär:	VLBalkNr	
Kandidat:	VLBalkNr	
Främmande:	VLNr	