Examinationsuppgift
C# & Datalagring med .Net

Pierre Lidholm

Innehållsförteckning

Komponenter i mitt program

Solution och projekt

Jag började med att skapa en solution som jag döpte till ”Examensuppgift – Pierre Lidholm”. När denna solution var skapad så skapade jag tre stycken projekt som jag lade in i min solution, vilket gjorde att jag har fyra stycken projekt i min solution. Dessa projekt heter Examensuppgift – Pierre Lidholm, DataAccess, DataInterface och UnitTest.

Jag skapade sedan dependencies mellan mina olika projekt.

Projektet Examensuppgift – Pierre Lidholm har dependencies till DataAccess och DataInterface, projektet DataAccess har en dependency till DataInterface, projektet DataInterface har ingen dependecy och projektet UnitTest har dependencies till DataInterface och Examensuppgift – Pierre Lidholm.

Projekten

De projekten jag skapade innehåller olika delar av mitt program, och varje projekt har en ett eget användningsområde i programmet.

Examensuppgift – Pierre Lidholm innehåller ett logik-lager som bekräftar att all data som matas in i databasen är korrekt.

DataAccess innehåller alla managers som jag använder för att ”prata” med databasen, och den innehåller även en klass som heter LibraryContext.

LibraryContext är en klass som skapar en förbindelse mellan databasen och mitt program.

DataInterface innehåller alla tabeller och interfaces som jag har skapat. I det projektet ligger även olika enum till de error codes som jag använder när jag bekräftar att datan som skickas in i min databas är korrekt.

UnitTest innehåller klasser som testar logik-lagret som testar att all data som matas in i programmet fungerar på ett korrekt sätt.

Tabeller

Jag började med att skapa tabellen Aisle. Jag började med denna tabell eftersom Aisle ska innehålla information om de olika gångarna som bokhyllorna ska stå i.

Jag skapade sedan tabellen Bookshelf. Bookshelf innehåller bokhyllorna som ska stå i varje gång.

Jag kopplade sedan ihop Aisle-tabellen och Bookshelf-tabellen genom att sätta en en-till-många-relation Aisle till Bookshelf. Detta gjorde jag därför att en Aisle kan innehålla flera Bookshelfs.

Tabellen jag skapade efter dessa två tabeller var Books. Books innehåller information om varje bok. I vilken gång boken ska stå i och i vilken bokhylla i den gången som boken ska stå i. Jag skapade en en-till-många-relation mellan tabellen Book och tabellen Bookshelf. Detta gjorde jag eftersom en bokhylla ska kunna innehålla flera olika böcker.

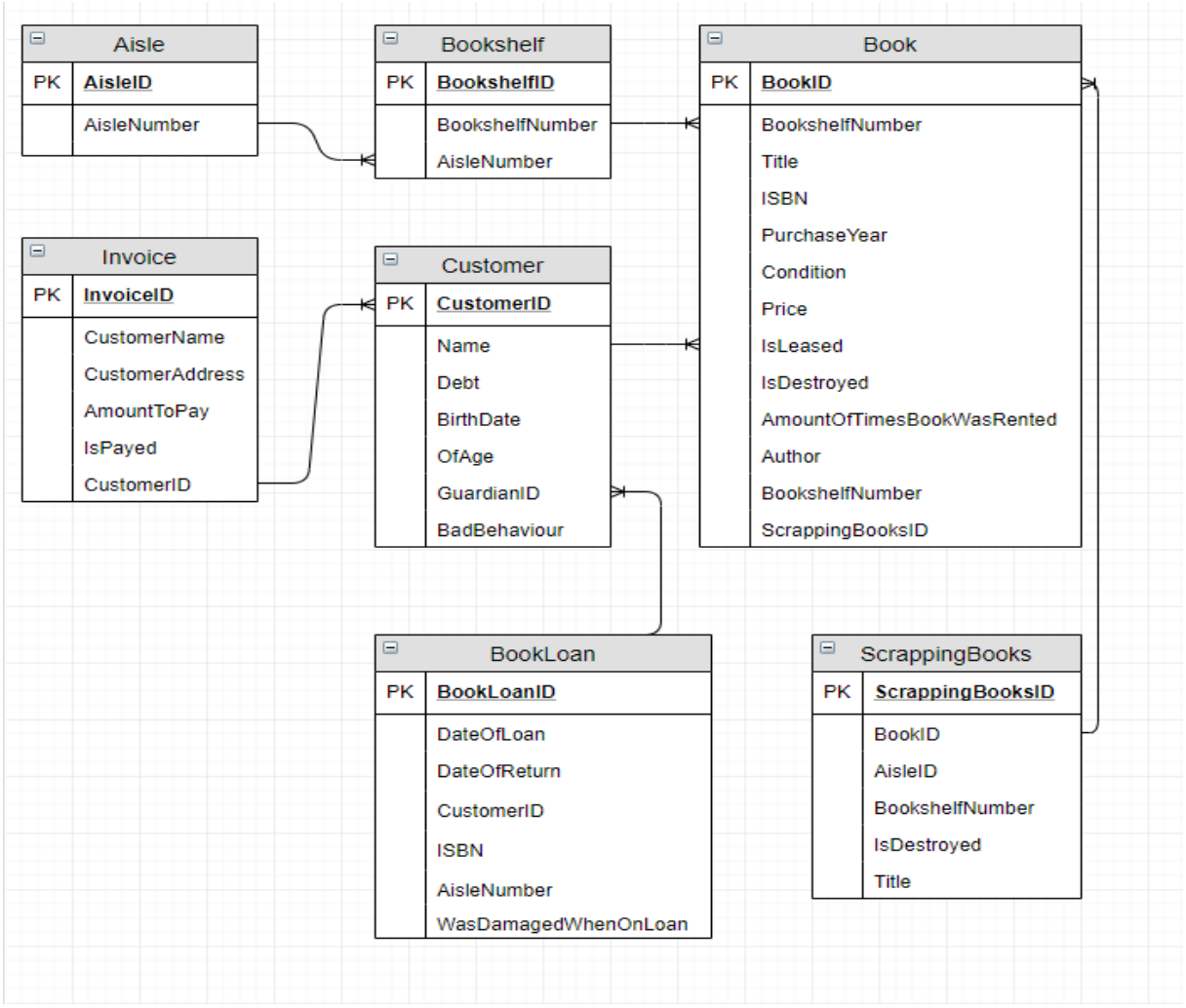
När dessa tabeller var skapade så skapade jag tabellen Customers. Customers innehåller all information om kunderna.

När Customers var skapad så skapade jag tabellen BookLoan. BookLoan är en tabell som innehåller information om alla boklånen som kunderna gör. BookLoan har en en-till-en-relation mellan de två tabellerna Customers och Books. Detta gjorde jag eftersom en kund kan göra ett lån av en bok per låning.

Efter Customers skapade jag tabellen BookLoans som innehåller information angående alla boklån som kunderna gör.

Efter att jag hade skapat BookLoans skapade jag tabellen Invoice. Invoice är en tabell som innehåller information om kundernas fakturor när de har lånat en bok. Den tabellen har en en-till-många-relation till tabellen Customers. Detta gjorde jag eftersom en kund kan ha flera fakturor utskickade till sig.

När tabellen Invoice var skapad så skapade jag tabellen ScrappingBooks. ScrappingBooks är en tabell som ska användas för att bibliotekarien ska kunna lägga in böcker i dåligt skick i en skrotningslista för att sedan klarmarkera denna lista och ta bort dessa från databasen. ScrappingBooks har en en-till-många-relation till tabellen Books.



Managers

Jag skapade sju olika managers i mitt projekt DataAccess. Dessa managers har som uppgift att prata med databasen genom att till exempel lägga till en bok eller ta bort en bok från databasen. Dessa sju managers döpte jag till AisleManager, BookLoanManager, BookManager, BookshelfManager, CustomerManager, InvoiceManager och ScrappingBooksManager.

AisleManager

Denna manager innehåller tre metoder.

En metod för att lägga till en gång i databasen, en metod för att ta bort en gång från databasen och en metod för att hämta en gång från databasen.

BookshelfManager

Denna manager innehåller fyra metoder.

En metod för att lägga till en bokhylla i databasen, en metod för att ta bort en bokhylla från databasen, en metod för att hämta en bokhylla från databasen och en metod för att flytta en bokhylla till en annan gång i databasen.

BookManager

Denna manager innehåller fem metoder.

En metod för att lägga till en bok i databasen, en metod för att ta bort en bok från databasen, en metod för att hämta en bok från databasen, en metod som returnerar alla böcker med dåligt skick från databasen och en metod som flyttar en bok från en gång till en annan gång eller en bokhylla till en annan bokhylla i databasen.

CustomerManager

Denna manager innehåller nio metoder.

En metod som lägger in en kund i databasen, en metod som lägger till en myndig person till en minderårig person i databasen, en metod som hämtar en myndig person från databasen, en metod som hämtar en minderårig person från databasen, en metod för att ta bort en kund från databasen, en metod som hämtar en kund från databasen, en metod som returnerar en lista av de fem sämsta kunderna från databasen och en metod som returnerar alla barn och deras vårdnadshavare som inte är dåliga kunder.

InvoiceManager

Denna manager innehåller fyra metoder.

En metod som skapar en faktura till en kund. En metod som hämtar vilken kund som ska faktureras och lägger in alla fakturor som tillhör den kunden i en lista. En metod som hämtar en faktura från databasen och en metod som sköter betalningen av en faktura i databasen.

BookLoanManager

Denna manager innehåller nio metoder.

En metod som lägger till ett boklån till en kund och lägger in detta i databasen, en metod som hämtar vilken kund som ska låna boken, en metod som hämtar vilken bok det är som ska lånas, en metod som hämtar ett boklån från databasen, en metod som sköter återlämningen av en bok i databasen, en metod som uppdaterar kundens status när en bok återlämnas, en metod som uppdaterar bokens status när denna bok återlämnas, en metod som hämtar ut alla böcker en kund har lånat.

BookScrappingManager

Denna manager innehåller fem metoder.

En metod som lägger in en dålig bok i databasen, en metod som hämtar alla dåliga böcker som inte är utlånade eller redan förstörda från databasen och returnerar de i en lista, en metod som hämtar denna lista från databasen, en metod tar bort dessa böcker från databasen och en metod som hämtar en lista av de böcker som ska förstöras från databasen.

Application Programming Interface

Jag skapade sju olika APIs i mitt projekt Examensuppgift – Pierre Lidholm. Dessa APIs döpte jag till AisleAPI, BookAPI, BookLoanAPI, BookshelfAPI, CustomerAPI, InvoiceAPI och ScrappingBooksAPI. Dessa API har som uppgift att se kontrollera att korrekt data läggs in i databasen.

AisleAPI

Detta API har två metoder.

En metod som kontrollerar den data som matats in när en gång ska skapas och en metod kontrollerar den data som matats in när en gång ska tas bort.

BookshelfAPI

Detta API har tre metoder.

En metod som kontrollerar den data som matats in när en bokhylla ska skapas, en metod som kontrollerar den data som matats in när en bokhylla ska flyttas och en metod som kontrollerar den data som matats in när en bokhylla ska tas bort.

BookAPI

Detta API har tre metoder.

En metod som kontrollerar den data som matats in när en bok ska läggas till i biblioteket, en metod som kontrollerar den data som matats in när en bok ska flyttas till en annan gång eller bokhylla i biblioteket och en metod som kontrollerar den data som matats in när en bokhylla ska tas bort från biblioteket.

CustomerAPI

Detta API har fyra metoder.

En metod som kontrollerar den data som matas in när en ny kund ska läggas till i bibliotekets databas, en metod som kontrollerar den data som matas in när en kund ska tas bort ur bibliotekets databas, en metod som kontrollerar den data som matas in när bibliotekarien vill hämta de fem sämsta kunderna och en metod som kontrollerar den data som matas in när biblioteket ska bjuda in barn till en barnkväll.

InvoiceAPI

Detta API har två metoder.

En metod som kontrollerar den data som matas in när en faktura ska skapas och en metod som kontrollerar den data som matas in när en faktura ska betalas.

BookLoanAPI

Detta API har två metoder.

En metod som kontrollerar den data som matas in när ett lån av en bok ska ske och en metod som kontrollerar den data som matas in när en bok lämnar tillbaka av en kund.

ScrappingBooksAPI

Detta API har två metoder.

En metod som kontrollerar den data som matas in när en boklista med dåliga böcker ska skapas och en metod som kontrollerar den data som matas in när en boklista med dåliga böcker ska tas bort.

Logiska uträkningar

Jag skapade tre klasser som tar hand om de logiska operationerna i programmet. Dessa logiska operationer är när en kostnad för en kund ska räknas ut, en logisk operation när ett datum ska räknas ut och göras om till en DateTime-variabel och en klass som sköter den uträknings som krävs när ett ISBN ska bekräftas.

Dessa klasser döpte jag till CalculateExtraCosts, DateCalculator och ISBNConfirmer.

API-teser

Jag skapade ett eget projekt för alla olika tester som jag gjorde på API:erna och databasen.

Jag skapade totalt sju olika test-klasser. Dessa klasserna döpte jag till AisleAPITests, BookshelfAPITests, BookAPITests, CustomerAPITests, BookLoanAPITests, InvoiceAPITests och ScrappingBooksAPITests.

I dessa tester gjorde jag logiska tester för att se om API-klasserna fungerade som de ska och att rätt data matas in i databasen.

Byggordning av programmet

Jag började med att bygga tabellen som innehåller information om alla gångar i databasen. Sedan byggde jag ett interface till den managern som har hand om att skriva till och läsa gångarna från gångarna i databasen. När jag hade byggt detta så skapade jag en API har hand om den information som ska skickas in i managern som har hand om gångarna. Sedan byggde jag alla tester av API:n till gångarna.

När detta var färdigt och testerna fungerade så skapade jag en tabell för alla bokhyllor i biblioteket. När denna tabell var klar skapade jag ett interface och en manager som har hand om informationen som läggs in och läses från databasen. När detta var klart byggde jag en API-klass som tar hand om den information som ska matas in i tabellen om bokhyllorna. När den API:n var klar skapade jag en test-klass för att testa så att logiken i API:n fungerar som den ska genom att stoppa felaktig inmatning från databasen.

När detta var färdigt och testerna fungerade skapade jag en tabell för alla böcker. När den tabellen var klar skapade jag ett interface och en manager som ärver från det interfacet. Denna manager har hand om informationen om böckerna som läggs in och läses från tabellen i databasen som innehåller informationen om alla böckerna. När denna manager var klar skapade jag ett API som bekräftar att informationen som matas in i databasen är korrekt. När denna API var klar skapade jag API-testerna för att kontrollera att API:n stoppar otillåten inmatning av information till tabellen om böckerna.

När detta var färdigt och alla tester fungerade som de skulle skapade jag en tabell för alla kunderna. När denna tabell var klarar skapade jag ett interface som jag ärvde av i min manager. Denna managern lägger in alla data och läser alla data som har lagts in i databasen. När den managern var klar skapade jag ett API som bekräftade och stoppade att felaktiga data läses eller skrivs in i databasen. När det API:t var klart så skapade jag API-tester för denna API.

När alla kundrelaterade klasser var färdiga så skapade jag en tabell som har hand om informationen angående alla boklån som ska ske. När denna tabell var färdigställd skapade jag ett interface och en manager som ärver från det interfacet. Denna manager har hand on all skrivning och läsning till och från tabellen angående boklånen i databasen. När denna manager var färdig så skapade jag ett API som bekräftar att rätt information matas in i managern som har hand om boklånen, och stoppar felaktig information som matas in. När detta API var färdigställt skapade jag API-tester av API:n där jag testade olika testfall.

När alla boklånsrelaterade klasser och funktioner var färdigställda och fungerade som de skulle skapade jag en tabell som innehåller information angående fakturorna som ska skickas ut till kunderna. Nästa steg jag tog var att skapa ett interface och en manager som ärver från detta interface. Managern som skapades har ansvaret att skriva till, och läsa från, tabellen Invoices i databasen. När denna manager var färdigställd skapade jag en API-klass som bekräftar att den informationen som matas in till managern är korrekt och att felaktig information stoppas innan den skickas in till managern. Efter detta skapade jag ett API-test där jag testar olika testfall och ser att API:n fungerar som den ska.

När alla dessa tabeller och managers var skapade så skapade jag tabellen ScrappingBooks. ScrappingBooks är en tabell som innehåller de böcker har dåligt skick och ska förstöras av bibliotekarien. När denna tabell var skapad så skapade jag ett interface och en manager som ärver från detta interface. Denna managers uppgift är att läsa från, och skriva till, tabellen ScrappingBooks. När denna manager var färdigställd så skapade jag en API-klass vars uppgift är att se till så att den information som matas in i managern är korrekt. API-klassen stoppar även felaktig information som matas in av bibliotekarien innan datan skickas till managern. När denna API-klass var färdigställd skapade jag en test-klass som testar olika inmatningsalternativ, felaktiga som korrekta, för att se till att API:n fungerar korrekt.

Jag valde att bygga min uppgift på detta sätt eftersom att jag anser att det gynnar kunden i första hand, men att det även gynnar biblioteket.