Affärssystem (Lab 5), Version 2

Dokumentation

Simon Åkerblom

6 May 2022

1. Antaganden

- Datan (försäljningar och artiklar) sparas i text-format.
- En vara är en instans av en artikel.
- Vid återköp kontrolleras att varan som ska returneras finns med i filen med loggade köp.

2. Översikt

Programmet består av ett fönster med två separata vyer (flikar) med olika funktionalitet: “Stock” (lagerhantering) och “Checkout“ (kassahantering).

I lagerhanteringsvyn kan man lägga till och ta bort artiklar, lägga order på varor av artiklar som är inlagda i systemet, samt visa topplista för mesta sålda varor och total försäljning för varje månad och/eller år. Produkt-kategorierna är uppdelade i separata vyer.

I kassahanteringsvyn kan man köpa och returnera varor. Det går att filtrera på produktkategori och en sökfunktion finns tillgänglig där man kan söka på en artikels namn. Man får även möjligheten att skriva ut ett kvitto på sitt köp.

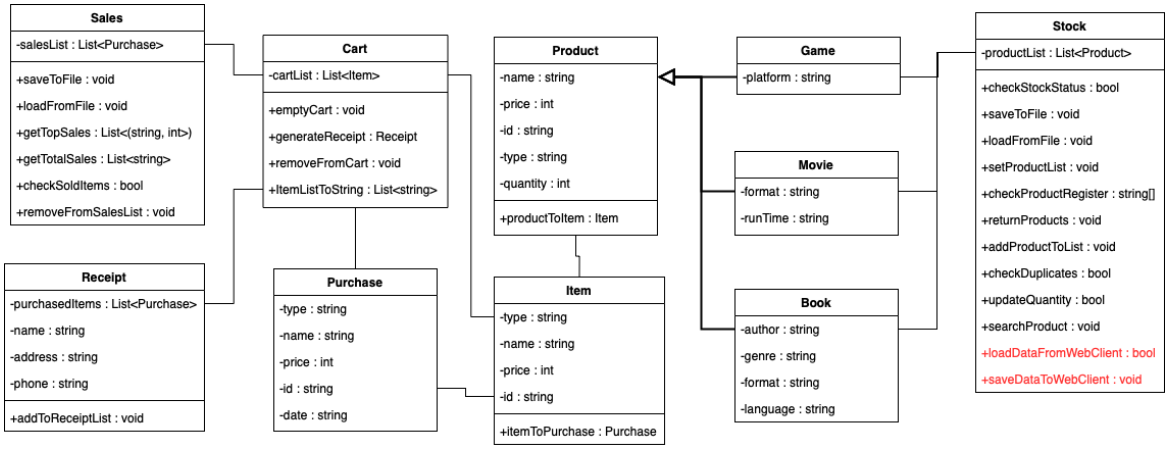
För att lägga till en ny vara fyller man i alla obligatoriska fält och klickar på “Add”-knappen. För att ta bort en vara väljer man en eller flera artiklar i listan och klickar på “Remove”-knappen. För att beställa varor av en viss artikel, välj först en eller flera artiklar i listan, ange antalet artiklar, och klicka sen på “Place order”-knappen. För att ta fram en topplista på mest sålda varor, välj först månad/år i rullgardinsmenyerna längst ned i fönstret, klicka sen på “Top Ten”-knappen. På samma vis funkar det för att ta fram statistik för total försäljning men då måste man välja både månad och år. Det finns en “Update”-knapp och en ”Sync”-knapp som synkroniserar varor med centrallagret via ett webb API.

För att lägga till en vara i kundkorgen, välj en artikel i listan och klicka på “Add To Cart”-knappen, alternativt dubbelklicka på artikeln. För att köpa en eller flera varor klicka på “Buy Product(s)”-knappen. Ett kvitto visas då där man ges möjligheten att skriva ut det. Det går att tömma eller ta bort enskilda varor från kundkorgen, det gör man med “Empty”- resp. “Remove”-knappen. För att returnera en vara, klicka på “Return”-knappen och fyll i artikelns ID. Om artikeln finns registrerad klicka på “Return”-knappen. För att filtrera på

produktkategori välj en kategori i rullgardinsmenyn och/eller ange namn på produkten i sökrutan precis bredvid.

3. Detaljerad beskrivning

MVC-mönstret har använts i ett försök att uppfylla “seperation of concern”. Det finns en Model, en View och en Controller. Nedan visas klasserna som ingår i Model och relationerna mellan dessa. En kort beskrivning av varje klass efterföljer.



Klassdiagram ritat med UML-notation som representerar Model-delen av programmet.

Stock:

Den här klassen hanterar alla sparade artiklar. Klassen tillhandahåller metoder för att söka i artikelregistret, lägga till och ta bort produkter, och filhantering.

productList - En lista som tar emot objekt av typen *Product*. Listan laddas från fil vid uppstart och sparas till fil vid stängning.

searchProduct :void - Tar emot en *String* från sökfältet och uppdaterar *BindingListSource* genom att anropa *setProductList()*.
loadFromFile :void - Hämtar alla artiklar som är sparade till fil, skapar *Product*-objekt, och lägger till i *productList*.
saveToFile :void - Sparar ned alla *Product*-objekt i *productList* till en .csv-fil.
***checkDuplicates* :bool** - Kontrollerar om *Id* redan används för en annan artikel i *productList*. Returnerar false om *Id* redan finns, annars true.
addProductToList :void - Lägger till en ny artikel i *productList*.
updateQuantity :bool - Uppdaterar antal (*Quantity*) för en specifik *Product* i *productList*.
setProductList :void - Ändrar datakällan kopplad till *GridView* i både lagerhantering och kassahantering.

checkStockStatus :bool - Kontrollerar att lagersaldo för en viss artikel inte är lika med noll och anropar *UpdateQuantity()* för att uppdatera status. Returnerar *false* om status är lika med noll, annars *true*.
checkProductRegister :string[] - Om en artikel vars *Id* är lika med metodens indata (string), returneras varans namn och typ, annars returneras tomma strängar. Anropas vid retur av vara.
returnProducts :void - "Lägger tillbaka" en tidigare köpt vara genom att ta bort *Purchase*-objektet i *salesList*. Uppdaterar därefter saldo genom att anropa *UpdateQuantity()*.
loadDataFromWebClient :bool - Synkroniserar artiklarnas pris och lagerstatus i *productList* med ett externt webb API. Om artikeln inte finns läggs den till i *productList*. Returner *false* om API:et returnerar ett felmeddelende, annars *true*.
saveDataToWebClient :void - Synkroniserar artiklarnas lagerstatus i *productList* med ett externt webb APi.

Stock är associerad med subklasserna till Product: *Game*, *Movie* och *Book*. *Product*-objekt är alltid en av dessa specialiseringar.

Product:

Representerar en artikel, antingen som *Game*, *Movie* eller *Book*.

name - Artikelns namn/titel.
price - Artikelns pris.
id - Artikelns unika id.
type - Artikelns typ: *Book*, *Movie* eller *Game*.
quantity - Representerar hur många av denna artikel som finns på lager.

productToItem :Item - Skapar ett *Item*-objekt och använder artikelns data som input. Returnerar det skapade *Item*-objektet.

Product-klassen är associerad med *Item*-klassen.

Game:

En subklass av *Product*.

platform - Spelets platform (Xbox, PC, Switch osv.).

Movie:

En subklass av *Product*.

format - Filmens format (DVD, Blue-ray osv.).
runTime - Filmens speltid.

Book:

En subklass av *Product*.

author - Bokens författare.
genre - Bokens genre (Skräck, Fantasy, Noir osv.).
format - Bokens format (Inbunden, pocket osv.)
language - Bokens språk.

Item:

En klass som representerar en artikel som någon lagt till kundkorgen.

type - Artikelns typ: *Book*, *Movie* eller *Game*.
name - Artikelns namn/titel.
price - Artikelns pris.
id - Artikelns unika id.

itemToPurchase :Purchase - Skapar ett *Purchase*-objekt och använder Item-objektets data som input. Returnerar det skapade *Purchase*-objektet.

Purchase:

En klass som representerar en köpt vara.

type - Artikelns typ: *Book*, *Movie* eller *Game*.
name - Artikelns namn/titel.
price - Artikelns pris.
id - Artikelns unika id.
date - Tid och datum vid köptillfälle.

Cart:

En klass som representerar de varor som lagts till i kundkorgen. Klassen tillhandahåller metoder för lägga till/ta bort från carList och generera kvitto vid köp.

cartList - En lista som tar emot objekt av typen *Item*.

emptyCart :void - Tar bort samtliga *Item*-objekt i *cartList*.
generateReceipt :Receipt - Genererar och returnerar ett kvitto och använder *Item*-objektens data som input. Returnerar det skapde *Receipt*-objektet.
removeFromCart :void - Tar bort ett specifikt *Item*-objekt från *cartList*.
ItemListToString :List<string> - Returnerar en lista med strängar som representerar samtliga *Item*-objekt i *cartList*. Visas i kundkorgen.

Sales:

Den här klassen hanterar alla försäljningar. Klassen tillhandahåller metoder för att ta fram top-10-listor / total försäljnings-listor och fil- och retur-hantering.

salesList - En lista som tar emot objekt av typen *Purchase*. Listan laddas från fil vid uppstart och sparas till fil vid stängning.

saveToFile :void - Sparar ned alla *Purchase*-objekt i *salesList* till en .csv-fil.
loadFromFile :void - Hämtar alla köp som är sparade till fil, skapar *Purchase*-objekt, och lägger till i *salesList*.
getTopSales :List<(string, int)> - Tar fram en topp-10-lista på mest sålda varor för en viss månad och/eller år. Returnerar en lista med artiklarnas namn och antalet sålda varor.
getTotalSales :List<string> - Tar fram en total försäljnings-lista där det totala antalet sålda varor för en viss månad och år beräknas. Returnerar en lista med antalet och vilka varor som sålts.
checkSoldItems :bool - Kontrollerar om *Id* finns med i *salesList*.
removeFromSalesList :void - Tar bort ett *Purchase*-objekt från *salesList* vid retur.

Receipt:

En klass som representerar den information som ska visas på kvittot. Genereras i *Cart*-objektet.

purchasedItems - En lista som tar emot objekt av typen *Item*.
addToReceiptList :void - Lägger till Item-objekt i *purchasedItems*.

---

4. Problem

Funktionalitet för utskrift var den enda delen av uppgiften som var helt främmande och tog ganska lång tid att lösa, då jag inte visste hur man går till väga för att programmera en utskrift. Det finns säkert en mängd sätt man kan göra det på och jag försökte först att “designa” utskriften själv, rad för rad, men det kändes för invecklat, så det slutade med att jag hittade en lösning där man enkelt kunde ta en skärmdump av just den komponent man var intresserad av och sen skriva ut den bilden. Fördelen med den lösningen är att kvittot ser ut exakt så som det visas på skärmen.

Separationen av vy och model har varit ett problem från start då det är knepigt att veta vilken logik som är okej i vyn och vilken som inte är det. Framförallt eftersom jag valt att använda mig av *BindingLists*. Det är svårt att veta exakt vad man får och inte får göra i vyn. Det är lätt hänt att

programmets logik slinker in i vyn när man har tillgång till hela "databasen" via *GridView*-komponenterna. Jag har försökt att tänka på det under tiden jag programmerat men det har ändå lett till en hel del omstrukturering.

Andra problem jag stött på har att göra med relationer/associationer mellan klasserna, framförallt i model. Att bestämma sig för vilka klasser som ska ha relation till vilka objekt har tagit ganska mycket tid. I vissa fall har jag valt att skicka med ett objekt i konstruktorn för att på så vis få tillgång till metoderna i det medskickade objektet, istället för göra om attribut/egenskaper till static. Vet inte vad som är att föredra men har för mig att man ska undvika att använda static om det inte verkligen är nödvändigt.

5. Sammanfattning

Det har varit intressant och lärorikt att bygga ett program från grunden med C# och .NET, och man har förstått vikten av att ha en design att utgå från. Ett enkelt klassdiagram (trots bristfällig) har underlättat en hel del under arbetets gång, framförallt eftersom man får en bra bild av hur stort/litet programmet man ska bygga är. Det var lättare att sätta igång och snabbt få till en grundstruktur utan att behöva tänka allt för mycket på detaljer.

Om man jämför klassdiagrammet före och efter implementering (se bilaga 1, 2) kan man se att mycket har förändrats, men en del av strukturen fanns där från början. En del klasser har andra namn för att göra designen mer lättförståelig. Vyn skiljer sig en hel del, antagligen för att jag inte funderade så mycket på gränssnitt och användarvänlighet innan implementering. Relationer mellan klasserna var svåra att förutse så de skiljer sig en hel del. I det nya klassdiagrammet är t.ex. *Controller* endast associerad med tre klasser i model.

När jag skulle byggda sökfunktionen och funktionerna för att ta fram listor behövde jag lära mig om *Language Integrated Query (LINQ)* för att på ett enkelt sätt hämta ut specifika objekt i listor. Jag har lärt mig använda *Ternary Operator,* vilket var smidigt, framförallt vid tilldelning av variabler och enklare villkorssatser. Visual Studio var enkelt att använda och väldigt smidigt vid omstrukturering och felsökning med *breakpoints*.

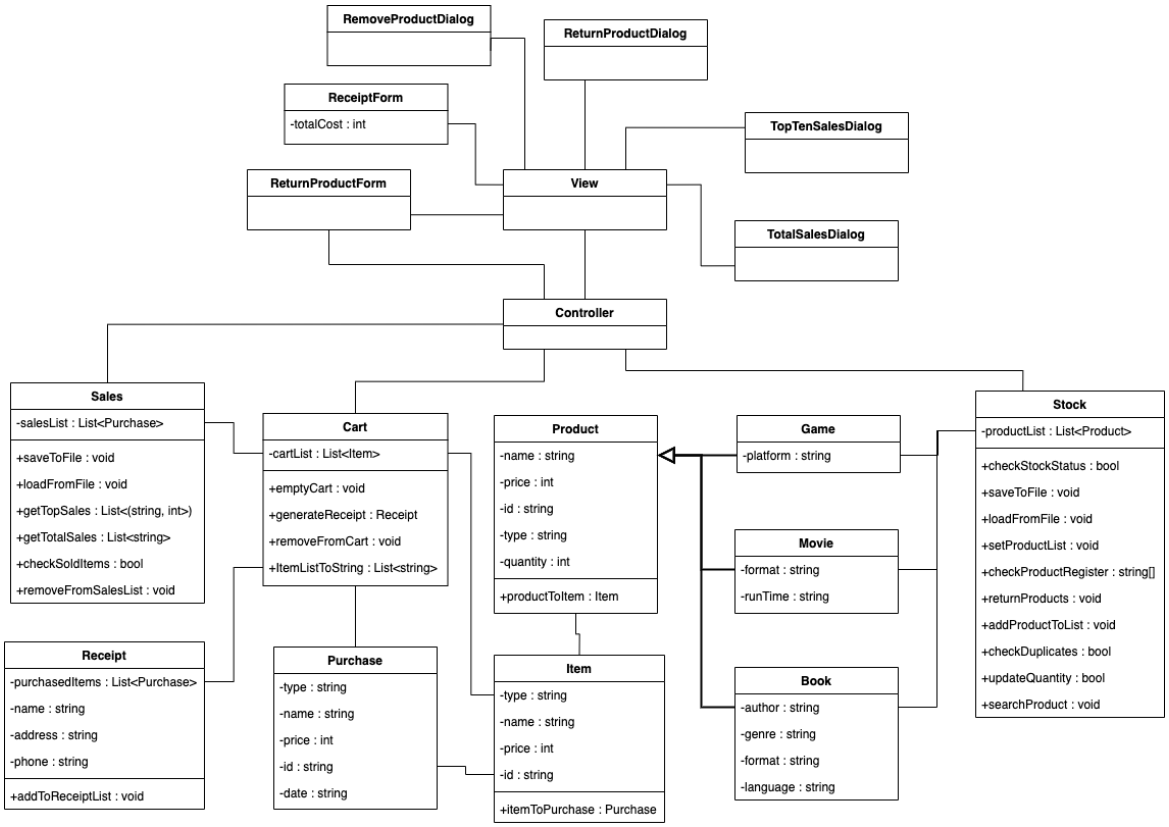
I en alternativ lösning hade jag kunnat lägga all filhantering i en separat klass, både för försäljning- och artikelregistret, vilket hade bidragit till

en tydligare separation av funktionalitet. Ett *Purchase*-objekt hade kunnat ha ett unikt försäljnings-ID och bestå av alla *Items* vid ett visst köptillfälle, istället för ett objekt för varje enskild *item*. Då hade man kunnat precisera och gruppera vilka varor som såldes när, och *retur*-funktionen hade blivit bättre. För att stödja att flera handlar varor samtidigt hade *Cart* kunnat skapas när första *Item*-objektet läggs till i kundkorgen för att sen tas bort när ett kvitto genererats.

Designen av det första klassdiagrammet tog ungefär 8 h och implementeringen satt jag med i ca. 2 veckor. Hur många timmar exakt är svårt att säga. Efter implementering gick det lättare och bygga ett nytt klassdiagram, ca. 4 h.

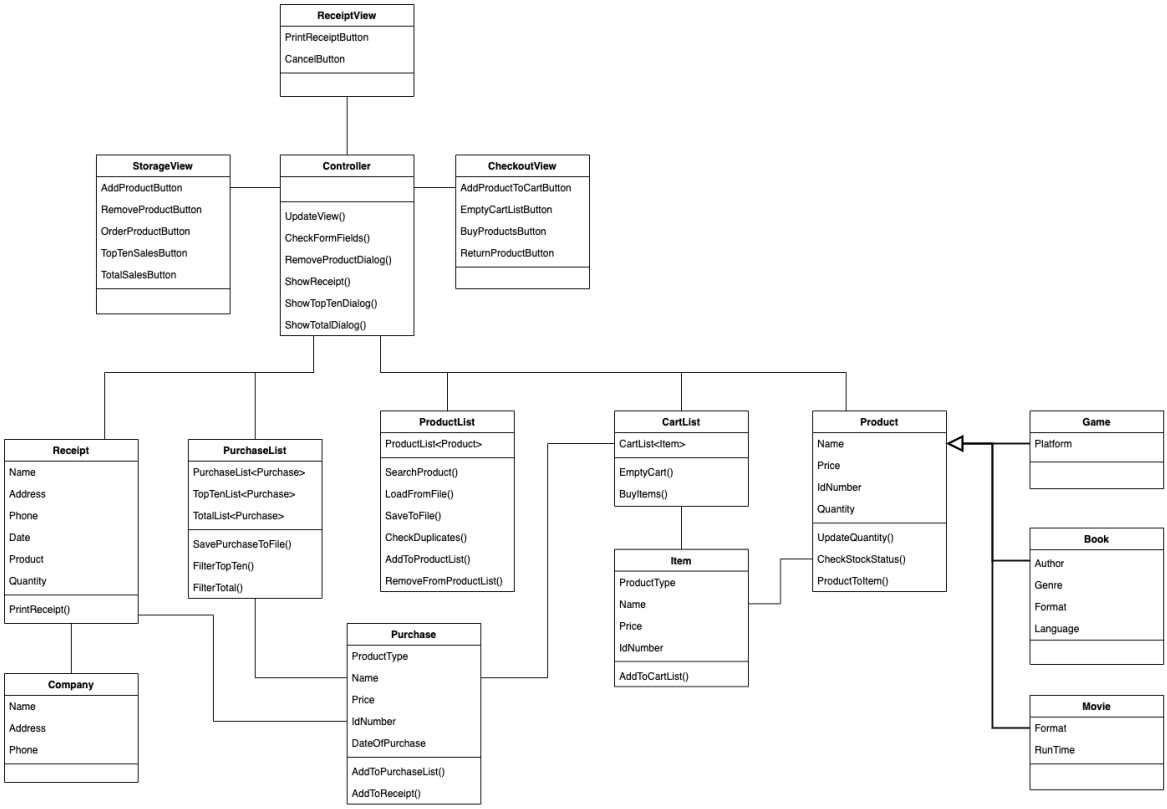
Inför laboration 5 var det nödvändigt att ändra hur produktkategorierna identifierades. I laboration 4 identifierades artiklarna via *Id*-egenskapen: “1-???” för *Book*, “2-???” för *Movie* och “3-???” för *Game*. Eftersom artiklarnas id i webb API:et inte hade samma format behövde de identiferas på annat vis. Det löstes genom att lägga till en ny egenskap i *Product,* döpt till *Type,* och som initieras varje gång en subklass till *Product* (*Book*, *Movie*, *Game*) skapas. Så istället för att undersöka objektens *Id* för att avgöra produktkateogorin, undersöks istället objektens *Type*-attribut.

Bilaga 1.



UML klassdiagram efter implementering

Bilaga 2.



UML klassdiagram före implementering