# Labbrapport 4: Oskar Deubler

# 1. Antaganden

1. Att filen ska sparas när programmet stängs av, antas vara okej att filen sparas vid flera tillfällen under programmet gångs, vilket i sin tur indirekt leder till att det är sparat när programmet stängs av.
2. Filen behöver bara öppnas i början av programmet.
3. I lagervyn behöver bara grundläggande information om varje produkt visas, för att hålla listan simpel.
4. Normalt användande av maxvärden, d.v.s. inget skydd för int overflow behövs.
5. Ingen storlek på programfönstret har angetts, därav antas det vara fritt val på storlek eller en storlek som passar för ändamålet.

# 2. Översikt

Programmet hanterar produkter i ett kassa- och lagersystem. I lagersystemet finns funktionalitet för att Lägga till, ta bort och registrera in-leveranser.

När programmet startar visas först en fil-väljarruta. Härifrån kan användaren leta upp en .csv-fil att läsa in. Detta kräver att .csv-filen har rätt format. Använd den fil som ligger i projektmappen. När en fil är vald, laddas produkter in i listan.

För att navigera mellan Kassa och Lager finns en navigeringsmeny längst upp på programmets fönster.

## **Lagervyn**

För att skapa en ny produkt, tryck på ”New Product”-knappen. 4 fält presenteras, 1 flervalsbox, och tre inmatningsfält. Fälten med en asterix (”Product”, ”Name” och ”Price”) måste fyllas i för att skapa en produkt, resten kan lämnas tomma. När en produkt väljs i ”Product”, dyker produktspecifika fält upp som kan fyllas i. När önskade fält är ifyllda, tryck på ”Add” för att lägga till. Här sker viss validering av fält som exempelvis kräver en specifik typ av inmatning. Vid fel dyker felmeddelanden upp och beskriver vad som är fel och vart.

För att se mer info om eller ta bort en produkt, klicka på en produkt i listan. En dialogruta visas upp, med mer information om den specifika produkten. I dialogrutan finns även en knapp för att ta bort produkten, och en knapp för att gå tillbaka. En varningsruta kommer dyka upp som frågar användaren om den verkligen vill ta bort produkten. Ifall produkten har kvantitet kvar på lager nämn det i varningsrutan.

För att registrera en in-leverans, börja med att trycka på Delivery-knappen. En till tom lista dyker upp till höger om produktlistan. Genom att klicka på en produkt i produktlistan till vänster, tillkommer den klickade produkten i leveranslistan. Användaren kan då trycka i alla de produkter som finns med på leveransen, sedan fylla i kvantitet i boxarna som finns under ”Qty” i lereranslistan. När användaren är klar, trycker den på ”Done”-knappen för att validera inmatningen och slutföra leveransen, varpå produktlistan uppdateras. Vid fel i inmatningen av kvanitet i leveranslistan, visas felmeddelanden när ”Done”-knappen trycks in. Därifrån kan användaren läsa för vilket product-id inmatningen av kvantiteten inte godkänns. Trycker användaren på ”Cancel”-knappen över leveranslistan rensas leveransen och listan göms.

## **Kassavyn**

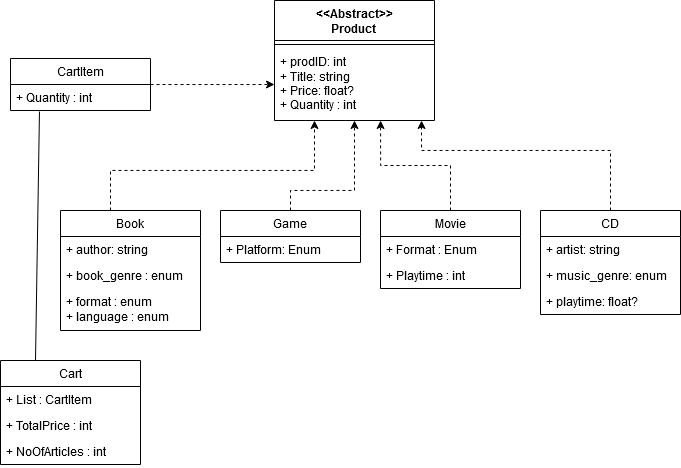
I kassavyn presenteras 4 listor. 3 av listorna befinner sig på 3 rader i den vänstra kolumnen, och är fyllda med Books, Movies och Videogames, de tre subklasserna till Product. Här visas all information upp för varje produkt. Den fjärde listan finns i höger kolumn, och representerar kundkorgen.

Ett köp påbörjas genom att trycka på en produkt i någon av de tre listor med produkter. Hela systemet för ett köp liknar det vid en inleverans. Skillnaden är att en ruta med det totala priset visas upp, bredvid två knappar för ”Check out” och ”Clear”. Priset uppdateras baserat på användarens input i ”Qty”. När alla produkter är inlagda i kundkorgen kan användaren trycka på ”Check out” för att färdigställa köpet. En dialogruta dyker upp för att säkerställa att en simulerad betalning görs. Därefter uppdaterar programmet listorna och dess UI där produkternas antal eventuellt ändras.

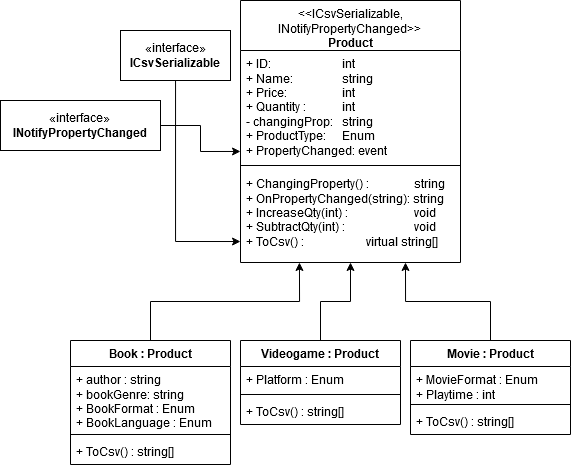
Om en produkt har 0 antal på lager så går det inte att lägga till de i kundkorgen.

# 3. Detaljerad beskrivning

## Klassdiagram



Figur 1. Klassdiagram vid planering av labben.



Figur 2. Klassdiagram efter implementering.

### **Kommentarer om klassdiagrammen**

Vid planeringen tycktes en ”abstract” klass för superklassen Product vara lämplig. Så blev inte fallet då för många metoder hade behövts implementeras flera gånger om för varje subklass. Därav dras nyttan av att ärva metoder direkt från superklassen för att slippa upprepad kod. Det slagkraftiga i att använda en superklass är att kunna lägga alla produkter i en och samma lista.

Idén om klasserna ”CartItem” och ”Cart” slängdes då grunden till kundkorgsfunktionaliteten redan implementerats när funktionaliteten för inleverans implementerades. För att påskynda projektets fortskridande återanvändes stora delar av existerande kod.

# Implementering

Programmets uppbyggnad följer inget specifikt designmönster, något som blev beklagligt mot slutet av implementationen. Huvudfilen MainPage.xaml.cs innehåller nästan all kod och är därmed inte så lättläst. Vid projektets början kändes projektet inte så stort som det till slut blev, vilket var orsaken till att inget designmönster användes.

Klass Product:

Superklassen för produkter. Implementerar gränssnitten ICsvSerializable och INotifyPropertyChanged.

Variabler:

* int ID – heltal som representerar produktens ID.
* string Name – Produktens namn
* int Price – Produktens pris
* int Qty – Produktens antal i lager
* private string changingProp - Variabel som lagras för att behålla värdet i en TextBox som skapas dynamiskt och länkas med Binding.
* event PropertyChangedEventHandler PropertyChanged - Händelse som reagerar på förändring, del av gränssnittet INotifyPropertyChanged.

Metoder:

* public string ChangingProperty() – Ändrar värdet i changingProp vid OnPropertyChanged. Används också som en get-metod för changingProp.
* public void OnPropertyChanged(string propertyName) – Anropar PropertyChanged för att trigga lyssnare för OnPropertyChanged på en produkt.
* public void IncreaseQty(int amount) – Ökar Qty med amount.
* public void SubtractQty(int amount) – Subtraherar Qty med amount.
* Public vritual string[] ToCsv() – Returnerar fälten i Product som en array av strängar. ”virtual” ser till att metoden kan bli överskriven i subklasserna.

Subklasserna av Product (Book, Videogame, Movie) är simpla, där variablernas namn beskriver vad de är och används till. Dessutom innehåller de bara metoden ToCsv() som redan finns beskriver i Product. Klassdiagramet i Figur 2 kan användas för att få reda på det som behövs för de olika subklasserna. Därav kommer inte dessa klasser beskrivas mer här.

Klass MainPage:

Programmet hjärna där majoriteten av all kod ligger. Härifrån styrs allt från filhantering till att visa och gömma UI-komponenter.

Variabler:

* List<Product> masterProducts - Masterdata för produkter. Huvudlistan för produkter för systemet.
* List<Product> deliveryProducts – Lista för leverans av produkter.
* List<Book> bookList – Lista för Book.
* List<Movie> movieList – Lista för Movie.
* List<Videogame> videogameList – Lista för Videogame.
* List<Product> cartProducts – Lista för produkter i kundkorgen.
* StorageFile CSVFile – Referens till .csv-filen.
* private int id\_counter – används för att sätta id på nya produkter.

Databas och datahanteringsmetoder:

* public async void Page\_Loaded(object sender, RoutedEventArgs e) – Körs när programmet laddas. Skickar vidare programmet till openFile(). Kontrollerar att en CSVFile inte är null, varpå ReadCSVFile och populateList körs, annars körs ErrorDialog().
* private async Task openFile() – FileOpenPicker öppnas för att välja en .csv-fil. Variabeln CSVFile tilldelas en StorageFile via PickSingleFileAsync().
* public async Task<List<string[]>> ReadCSVFile(StorageFile file) – Läser file rad för rad och sparar i List<string[]> data. Eftersom .csv-filer separerar kolumner med ’;’, splittas raderna upp med .Split(’;’), för att få rena data i string[]-format.
* public void populateList(List<string[]> data) – tar data och använder den för att fylla masterProducts med produkter via parseCsvData(). Kör sedan updateMasterProductsList() och updateSubclassLists() för att uppdatera UI.
* Internal List<Product> parseCsvData(List<string[]> data) – Går igenom data och parserar raderna till rätt typ av Product via parseProductFromString(). Returnerar en lista av Product.
* private Product parseProductFromString(string[] line) – Identifierar vilken subklass av Product som line är, därefter parserar den resten av informationen, även subklasspecifika attribut. Returnerar den subklass av Product som specificeras för line.
* private async void WriteToFile() – Kör WriteToCsv(masterProducts) för att spara masterProducts till CSVFile.
* Internal async Task WriteToCsv(List<Product> products) – Skapar en StringBuilder för att bygga en sträng som kan skrivas till CSVFile. Tar varje Product i products och gör om dessa till en sträng i rätt format, via ToCsv(). Skriver sedan till CSVFile via FileIO.WriteTextAsync.

Lägg till produkt – metoder:

* private async void ShowAddProductDialog() – Startar AddProductDialog (Se separat klassbeskrivning). Resultatet från AddProductDialog hanteras, varav den nya produkten hämtas från addDialog.NewProduct. Ett ID sätts med id\_counter och kontrolleras med isIdUsedInMasterdata tills ett godkänt ID hittas. Därefter läggs den nya produkten in i masterProducts och updateMasterProductsList anropas.
* private void NewProduct\_Click(object sender, RoutedEventArgs e) – Klickevent för ”New Product”-knappen. Kör ShowAddProductDialog().

Ta bort produkt – metoder:

* private async void ProductList\_ItemClick(object sender, ItemClickEventArgs e) – Klickevent för produkter i ListView ProductList. Kör ProductInfoDialog (se separat klassbeskrivning). Baserad på produktens Qty körs RemoveProductDialog med olika meddelanden.
* private async Task RemoveProductDialog(Product clickedItem, string message) – Kör en MessageDialog som validerar att användaren vill ta bort en produkt. Två knappar, ”Remove” och ”Cancel” finns med. Om ”Remove” så klörs removeProductFromMasterdata(clickedItem).
* private void removeProductFromMasterdat(Product product) – Tar bort product från masterProducts, och uppdaterar sedan UI med updateMasterProductsList().

Registrera inleverans – metoder:

* private void DeliveryButton\_Click(object sender, RoutedEventArgs e) – Ändrar UI genom att visa DeliveryList och dess knappar. Ändrar ItemClick-händelse för ProductList till ProductList\_AddItemToDelivery.
* private void ProductList\_AddItemToDelivery(object sender, ItemClickEventArgs e) – Gör en ny referens av den klickade produkten och lägger den i deliveryProducts om den inte redan finns där. Uppdaterar UI med updateDelvieryProductsList().
* private void updateDeliveryProductsList() – Fyller ListView DeliveryList med produkterna i deliveryProducts.
* private void DeliveryList\_ItemClick(object sender, ItemClickEventArgs e) – Klickhändelse för produkter i DelvieryList. Tar bort produkten från DeliveryList och uppdaterar UI med updateDeliveryProductsList().
* private async void DoneButton\_Click(object sender, RoutedEventArgs e) – Klickevent för ”Done”-knappen. Anropar validateChangingProperties(delvieryProducts) för att validera input i TextBoxar. Går valideringen igenom anropas RegisterDelivery(), och gömmer UI med HideDeliveryUI().
* private async Task<bool> validateChangingProperties(List<Product> products) – ChangingProperty valideras genom att traversera products och validerar att ChangingProperty är en positiv int och inte heller tom. Vid fel i valideringen körs ErrorDialog med ett meddelande som beskriver vart felet är så användaren får info.
* private void RegisterDelivery() – Parserar alla ChangingProperty i deliveryProducts, hittar den matchande produkten i masterProducts och ökar Qty med .IncreaseQty(qty). Uppdaterar sedan UI med updateMasterProductsList().
* private void HideDeliveryUI() – Gömmer leveransdelens UI, och rensar delvieryProducts. Återställer ListView ProductList:s klickhändelse till ProductList\_Itemclick().
* private void CancelButton\_Click(object sender, RoutedEventArgs e) – Avbryter leveransen genom HideDeliveryUI() och ClearChangingProperties().
* private void ClearChangingProperties() – Traverserar masterProducts och sätter alla dess produkters ChangingProperty till string.Empty.
* private async Task ErrorDialog(string message) – Visar en MessageDialog, skapar en OK-knapp.

Kassa – metoder:

* private void updateSubclassLists() – Sorterar och fyller subklassernas listor (bookList, movieList, videogameList) med hjälp av masterProducts. Kör sedan PopulateSubclassListViews().
* Private void PopulateSubclassLists() – Uppdaterar suklassernas ListViews och fyller de med produkter.
* private void ListView\_ItemClick(object sender, ItemClickEventArgs e) – Klickhändelse när en produkt från subklasslistorna klickas. Lägger till produkten i cartProducts OM produkten inte redan är i cartProducts och dess Qty är över 0. Produktens ChangingProperty sätts till ”1”. Uppdaterar kundkorgen med updateCartList().
* private void updateCartList() – Uppdaterar kundkorgslistan och uppdaterar TextBlocket med totala priset.
* private void updateTotalPrice() – Traverserar cartProducts och räknar ihop dess totala pris baserat på .Price och .Qty för varje produkt. Ändrar priset i TotalPriceTextBox.
* private void CartList\_ItemClick(object sender, ItemClickEventArgs e) – Klickhändelse för ListView CartList. Tar bort en produkt ur cartProducts vid klick. Uppdaterar UI via updateCartList().
* private void TextBox\_TextChanged(object sender, TextChangedEventArgs e) – TextChange-händelse för alla TextBoxar i kundkorgen. Reagerar om texten inte längre går att parsera till int. Sätter en bakgrundfärg för att visa vart det är fel. Om det går att parsera, uppdateras TotalPrice med updateTotalPrice.
* private async void CheckOutButton\_Click(object sender, RoutedEventArgs e) – Klickhändelse för ”Check out”-knappen. Kör en valideringsmetod validateCart(), går den igenom körs MakePurchaseDialog().
* private async Task<bool> validateCart() – Validerar alla ChangingProperty i cartProduts genom validatechangingProperties(). Kontrollerar även så att inte antalet av en produkt som sälj överskrider dess saldo (Qty). Returnerar en bool isValid.
* private async Task MakePurchaseDialog() – Kontrolldialog när en köp ska genomföras. Simulerar även mottagandet av betalning. Om knappen ”Payment Received” trycks in körs makePurchase().
* private void makePurchase() – Registrerar köpet på så vis att det ändrar saldo för produkterna i cartProducts. Därefter rensas cartProducts och alla berörda listor uppdateras.
* private void ClearCartButton\_Click(object sender, RoutedEventArgs e) – Tömmer cartProducts och dess ChangingProperty nollställs.

Övriga metoder:

* private void updateMasterProductsList() – Lägger masterProducts i en ListView ProductList. Kör även WriteToFile() för att spara.
* private void NavigationView\_SelectionChanged(NavigationView sender, NavigationViewSelectionChangedEventArgs args) – Ändrar mellan Lagervy och Kassavy.

Klass ProductInfoDialog: Avhåller en ContentDialog som visar information om produkten, specifik information baserat på vilken produkt det är. Metoderna uppdaterar endast dialogens UI.

Klass AddProductDialog: Avhåller en ContentDialog för att lägga till en ny produkt. Här ingår valideringsfunktioner och funktioner för att ändra dialogens UI baserat på vilket typ av produkt som ska läggas till.

Variabler:

* Product NewProduct – Innehåller den nya produkten som skapas.

Metoder:

* private void ContentDialog\_PrimaryButtonClick(ContentDialog sender, ContentDialogButtonClickEventArgs args) – Kör !isFormValid() i en if-sats. Om true, så avbryts args.Cancel, vilket gör att dialogen hålls öppen vid valideringsfel. Annars tilldelas NewProduct = ExtractProductInfo().
* private bool isFormValid() – Kontrollerar alla nödvändiga inputfält och om dess värden är av rätt typer.
* private Product ExtractProductInfo() – Typ av produkt bestämmer vilken data som ska extraheras från respektive TextBox. Returnerar Product, dock är detta en polymorf av den faktiska subklassen som returneras.
* Book ExtractBookInfo(), Movie ExtractMovieInfo(), Videogame ExtractVideogameInfo() – Extraherar först de delade attributen (ID, Name, Price, Qty) via GetCommonProductInfo() och sedan produkttypsspecifika attribut från respektive subklass.
* private void ProductComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e) – Visar de textboxar som hör till vald produkttyp, och gömmer de andra.

# 4. Problem

Det var svårt att, på ett simpelt sätt, kunna komma åt dynamiskt applicerade TextBoxar i en ListView eftersom de använder sig utav ListItem.DataTemplate. Då kan man inte ge de ett Name=”namn” som gör att de går att manipulera från .cs-filen. Därav byggdes lösningen med OnChangeProperty i Product-klassen. Som tur var kunde denna lösning senare användas inte bara i leverans-funktionen men också i kassa-funktionen.

# 5. Sammanfattning

Det var svårt att inse hur stort projekt detta verkligen är. Vid första anblick kändes det som ett projekt med ganska grundläggande funktioner men när allt ska bakas ihop var det komplicerat.

Min lösning är inte vacker, och verkligen inte en optimal lösning. Det finns cirka noll struktur i koden, antagligen agerar jag ett orakel för den eftersom jag skapat den. Jag hade valt att göra en lösning med MVC eller MVVM för att strukturera koden och få lite separation of concern.

Jag ville från början implementera med ett factory-designmönster men insåg att det skulle ta längre tid att lära sig hur det funkar rent kodmässigt än det skulle vara effektivt för detta projekt.

I övrigt har jag lärt mig att det inte är så bra med databaser i form av .csv-filer.

Jag började rätt tidigt med denna laboration, vilket var bra, för hade jag inte gjort det hade jag nog behövt spendera fler helger.