Seminarium 3

Objekt-Orienterad Design, IV1350

Daniel Westerlund
daweste@kth.se

2019-06-02

Innehållsförteckning

1 Introduktion

Uppgiften är att programmera ett program som är baserat på de tidigare seminarium lösningarna med fokus på flödet, designen och ett startscenario. Klassen View får hårdkodas med anrop till controllern. Dom externa systemen får utelämnas, dock skall programmet implementeras så att den kan anropa dem.

Jag har arbetat själv.

2 Metod

Använde resultaten ifrån de tidigare seminariumen som grund och började med att koda de klasser som är väsentligast för att kunna genomföra en försäljning, dvs klassen Sale, ProductList och Product. Därefter skapades en testklass, testAddProduct, som namnet antyder så försöker den addera en produkt till ett köp. addProduct tar i mot ett argument, en ean-kod. Det testfallen som valdes var:

- Ean-kod på en produkt som finns i klassen ExternalInventory.
- Ean-kod på en produkt som inte finns i klassen ExternalInventory.
- Lägga till en produkt med samma ean-kod flera gånger.
- Lägga till flera produkter med enbart olika ean-koder.
- Lägga till flera produkter med både ean-koder som redan finns och nya.

I klassen CurrentDiscounts som tar i mot ett ProductList-objekt och retunerar ett uppdaterat ProductList-objekt med rabatt avdragen, finns alla aktuella rabatter, dessa rabatter är hårdkodade. Det skapades även två testklasser som involverar CurrentDiscounts-klassen, TestAddDiscountToToTPrice och TestAllDiscountsAndVAT. Det finns två hårdkodade rabatter i CurrentDiscounts, 3cheeseForPriceOf2, halfPriceOnProduct(enda produkten med halva priset är beer, eankod 1002), även rabatten från en DiscountClass testas. De testfallen som skapades i TestAddDiscountToToTPrice var:

- Skapa ett product-objekt “beer”, och addera 7 sånna till nuvarande köp, därefter jämföra så att priset stämmer överens med det förväntade priset.
- Skapa ett product-objekt ”cheese”, och addera 8 sådana till nuvarande köp, därefter jämföra så att priset stämmer överens med det förväntade priset.
- Adderade en godtyckligt mängd av produkter och antal av dessa, samt med studentrabatt, därefter jämföra så att priset stämmer överens med det förväntade priset.
- Adderade en godtyckligt mängd av produkter och antal av dessa, utan någon medlemsrabatt, därefter jämföra så att priset stämmer överens med det förväntade priset.

De testfallen som skapdes i TestAllDiscountsAndVAT var:

- En testmetod med godtyckligt antal produkt och antal av dessa, därefter drogs alla rabatter av, inkluderat studentrabatt, därefter adderades VAT till det totala priset och jämfördes med det förväntade priset.

På dom externa systemen, ExternalAccounting, ExternalInvenotry och RecipePrinter implementerades några hårdkodade system för att likna en verklig miljö.

3 Resultat

Resultatet av kodningen kan ses på: https://github.com/Mrw17/IV1350_Sem3.
Fig 1 visar en testkörning utav programmet med hårdkodade anrop.

```
Very Cool shop Coolstreet 123 12345 cooltown


Current inventory (product, quantity)
{cheese 20.0=15, beer 15.0=30, scratch 25.0=100, Cigarett 45.0=50}
Current cashregister balance: 1000.0



***Cashier starts a new sale***
***Cashier scanning product(cheese)***
Show total price 20.0
***Cashier scanning product(cheese)***
Show total price 40.0
***Cashier scanning product(cheese)***
Show total price 60.0
***Cashier scanning product(cheese)***
Show total price 80.0
***Cashier scanning product(scratch)***
Show total price 105.0
***Cashier scanning product(beer)***
Show total price 120.0
***Cashier scanning product(beer)***
Show total price 135.0
***Cashier scanning product(cigarette)***
Show total price 180.0
***Cashier scanning product with eancode that doesnt exists in inventory***
Show total price 180.0
***No more products to scan***


Show total price with discount: 145.0
Show total price with discount + VAT: 170.25
***Customer  approves***
***Cashier approves***
Customer pays 100.0
You haven't payed enough, still missing:  70.25
Customer pays 100.0
You have  payed enough, change is:  29.75

*****Recipe*****
Date: 2019-06-02 01:53:25
Very Cool shop Coolstreet 123 12345 cooltown
cheese 20.0 x 4
scratch 25.0 x 1
beer 15.0 x 2
Cigarett 45.0 x 1

Total price: 170.25
Total Discount 9.75
Total VAT: 25.25

Paid: 200.0
Change: -29.75
*****Recipe*****



Current inventory (product, quantity)
{cheese 20.0=11, beer 15.0=28, scratch 25.0=99, Cigarett 45.0=49}
```

Figur 1- Testkörning utav programmet med hårdkodade anrop

4 Diskussion

Målet har varit att försöka skriva metoder som är enkla att förstå för en utomstående utvecklare. När man själv har suttit med koden i timme efter timme upplever jag själv att den relativt enkel, dock svårt att ta ställning till om fallet verkligen är så. Det finns lite duplicerad kod, som t.ex. när försäljaren och kunden ska acceptera köpet, som ändras statusen på deras variabler i klassen ApprovedSale, och därefter anropar en metod som kontrollerar fall båda har godkänt för att spara ner tiden när båda har godkänt, vilket gör att det blir lite duplicerande av kod och den metoden kommer troligtvis att ändras till seminarium 4, kanske rent av är onödigt med att försäljaren måste godkänna, skulle kunna vara en verksamhetsregel att om försäljaren godkänner köpet, så implicerar det att båda parter har godkänt.
Enhetstesterna är få, men testar stora väsentliga delar utav programmet. Egentligen kan det vara bättre att försöka bryta upp dom ännu mera, men känns som man nästan bara testar setters/getters då, vilket känns onödigt.
Är fortfarande lite osäker på om jag hanterar DTO:s på rätt sätt. T.ex. när objektet Recipe ska skapas så instansierars ett SaleDTO-objekt som tar ett Store-objekt och ett Sale-objekt som argument. Då spar jag ner den datan jag anser jag att behöver i lokala variabler och sen skapar getter-metoder för att få tillgång till den, känslan är att man skulle spara en del kod(ev. exekveringstid om man har stort program/stora DTO klasser?) om man i en getter-metoden returnerade t.ex. Store.getStore() direkt, istället för att hämta upp den, spara den till en variabel. Nu känns det som att det blir en den ”dubbeljobb”.
En del va programmet som jag inte är nöjd med än Product-klassen, egentligen borde den endast representera en produkt och attributer som amountSold och discount borde placerats i ProductList-klassen för att skapa tydligare strukturer. Tycker även att lösningen för rabatter blev onödigt komplicerad och statisk, kom inte på något bra sätt att skapa ett flexibelt rabattsystem.