

Seminarium 4

Niloofar Rahmani, nrahmani@kth.se
June 9, 2021

Innehåll

1 Introduktion	3
2 Metod	4
3 Resultat	5
4 Diskussion	6

1 Introduktion

Seminarium 4 handlar om undantag, varför man ska använda de och hur man ska skapa dem. Detta hjälper helt enkelt att kunna genomföra ett färdigt objekt orienterat program. För att få det nödvändiga kunskap för att genomföra uppgifter har jag använt kurslitteraturen och föreläsningar. Dessutom har mycket information fåtts av Google sökning och även äldre studenter har hjälpt för att förstå visa begrepp. Jag har samarbetat med Alva Ols och all kod är skrivna tillsammans.

2 Metod

Vi började som vanlig att kolla på föreläsningar och läsa kurslitteraturen för att förstå vad är målet och hur vi ska genomföra det. Sedan började vi med att kategorisera checked och unchecked undantag.

Task 1

Vi började med att skapa vår första checked undantag som skulle uppstå när det efterfrågan varan inte hittas. Det kallade vi för "ItemNotFoundException" som står under paketet dbHandler vilken kan även kallas för integration. I denna undantagklass skrev vi kod som skulle hitta vilken "item identifier" det är som inte finns och returnera den.

Vår unchecked undantag som skulle uppstå när det är något fel med själva programmet, kallade vi för "DatabaseErrorException" och finns också under dbHandler. Vi använder båda dessa undantag under inventorySystem klassen under dbHandler där det finns metod findItem. Det är då man antingen hittar varan eller ItemNotFoundException undantaget uppstår.

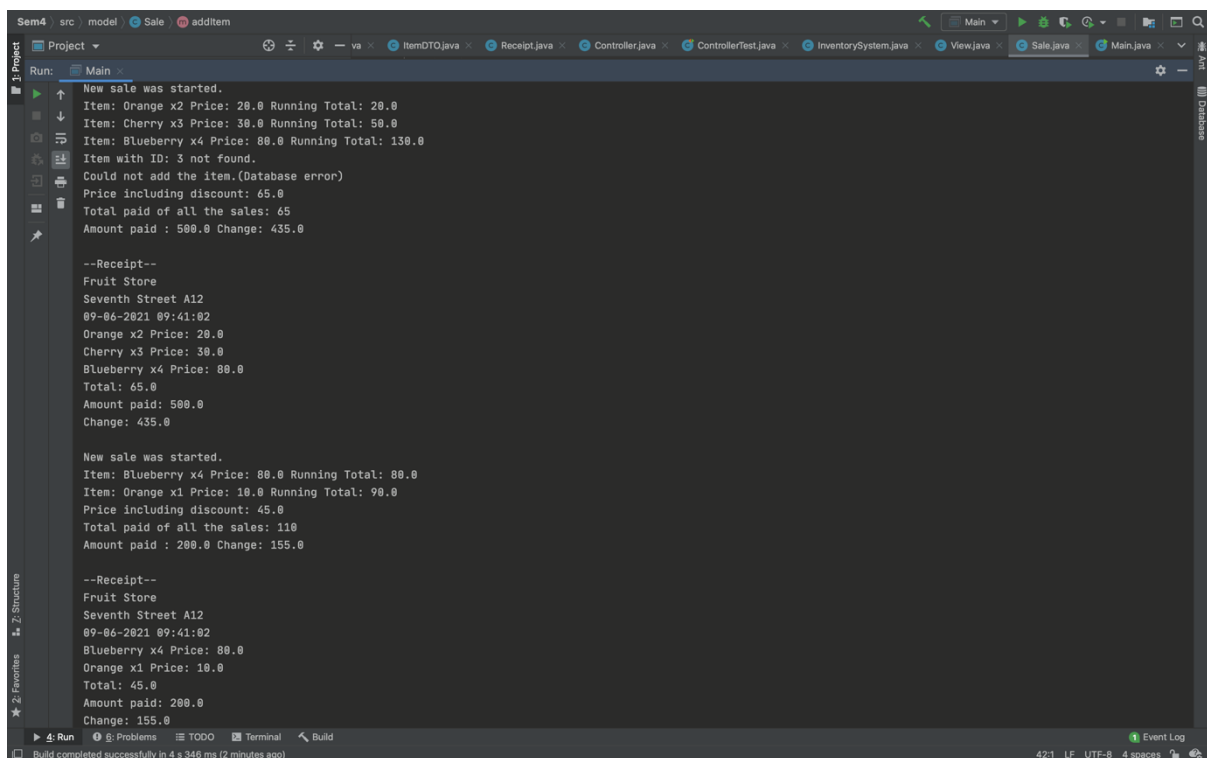
Task 2

Task 2 handlade om observers. Vi följde instruktioner och skapade en ny klass som skulle helt enkelt räkna totala belopp och "revenue". Observer specificera kommunikationen mellan objekter. Detta förklaras mer under diskussion.

3 Resultat

Länk till kod : <https://github.com/Niloo-r/IV-1350/tree/main/Sem4>

Efter vår godkänd på seminarium två fick vi lite mer självförtroende och tänkte uppgradera vår kod och utmana oss inför seminarium 4. Figur 3.1 visar det slutliga version av kvittot som skrivs ut men även visar att "exceptions" fungerar. Två sale sker i denna händelse som visas i varsin "New sale" och sedan skrivs ut varsitt kvitto till dem.



```
Sem4 src model Sale additem
Project
Run: Main
New sale was started.
Item: Orange x2 Price: 20.0 Running Total: 20.0
Item: Cherry x3 Price: 30.0 Running Total: 50.0
Item: Blueberry x4 Price: 80.0 Running Total: 130.0
Item with ID: 3 not found.
Could not add the item.(Database error)
Price including discount: 65.0
Total paid of all the sales: 65
Amount paid : 500.0 Change: 435.0

--Receipt--
Fruit Store
Seventh Street A12
09-06-2021 09:41:02
Orange x2 Price: 20.0
Cherry x3 Price: 30.0
Blueberry x4 Price: 80.0
Total: 65.0
Amount paid: 500.0
Change: 435.0

New sale was started.
Item: Blueberry x4 Price: 80.0 Running Total: 80.0
Item: Orange x1 Price: 10.0 Running Total: 90.0
Price including discount: 45.0
Total paid of all the sales: 110
Amount paid : 200.0 Change: 155.0

--Receipt--
Fruit Store
Seventh Street A12
09-06-2021 09:41:02
Blueberry x4 Price: 80.0
Orange x1 Price: 10.0
Total: 45.0
Amount paid: 200.0
Change: 155.0
Build completed successfully in 4 s 340 ms (2 minutes ago)
```

Figur 3.1 kvittot som skrivs ut samt det som står på skärmen vid två sale.

På första sale visas det hur det ser ut när en error i systemet sker som gör att varan inte kan läggas till. Däremot skickas det även meddelande om varan inte finns i inventory system. Vi valde att identifiera varorna med ett nummer vilket är precis som barcode.

4 Diskussion

Vi har tre exceptions : `CouldNotAddItem` (kastas när ett problem eller error uppstår medans man lägger till varorna till systemet d.v.s scanna) , `DatabaseError` (kastas när ett problem med databasen och systemet uppstår och kassören får meddelande om det på skärmen) och `ItemNotFound` (kastas när sökning för en vara har inte haft några resultat, d.v.s varan inte finns.)

Vi har skapat "`CouldNotAddItem`" under controller paketet. Vårt resonemang till detta var att processen av att lägga till varor sker i controller klassen och det är då ett undantag som misslyckas med att lägga till varan ska kastas. Däremot finns de andra undantag under `dbHandler` som även kallas integration. Logiken bakom det var att "`DatabaseError`" handlar ju om ett fel i programmet och programmerare ska kunna fixa det. "`ItemNotFound`" var också rimligt att vara under `dbHandler` eftersom det är inventory system som har koll på vilka varor som finns.

Vi har kastat dessa undantag i main och de "catchas" eller fångas i view för att visa det på skärmen så kassören kan se. Förutom det kastas visa under controller och inventory system också där det känns rimligt. Till exempel under inventory system, det som koden gör är att kolla om det efterfrågat varan finns och returnera den, annars kastas undantag som meddelar om att varan inte hittades.

`SaleObserver` som jag har förstått med hjälp och samaebete av `totalRevenue` ska ha koll på det totala betalning och inkomst som har skett. Detta kan då vara under en dag eller under längre perioder till exempel en vecka, månad eller år. Vi hade det lite svårt med den delen men det kan även var för att vi hade lite brist på tid.

Dessa tre undantag kan man se tydligt i figur 3.1. Vi är lite osäkra om vi har skapat dessa undantag under rätt och rimligt paket. Vi är även lite osäkra om vi har kastat och fångat de i rätt ställe.

Som funktioner inbyggda i java har vi använt oss av `java.lang.Math` för att beräkna växel som kunden ska få tillbaka. Även fast det var bara några ändringar och klasser som skulle göras i detta seminarium, kändes det väldigt mycket då vi inte riktigt förstod hur vi skulle göra, vad vi skulle få ut och hur alla errors som vi fick skulle fixas. Vi tycker vårt slutliga kvitto ser bra ut och vi kände inte att vi skulle ändra något.