Multifabriken – Reflektioner – Johannes Jakobsson

Kodbeskrivning

Eftersom ni läsare redan vet hur själva användandet av multifabriken fungerar tänkte jag att vi går rakt på sak och berättar hur jag har byggt upp min kod. Jag valde att skapa en klass för varje produkt, samt en klass för menyn där jag bygger stora delar av applikationen.

Allt börjar med att det skapas ett nytt objekt av klassen *Menu* och metoden *RunMenu* anropas, detta görs i klassen *Program* under metoden *Main*. *Menu* skapar vid det här laget fyra nya listor; *car-*, *candy-*, *lace-* och *tofuList,* en lista för varje produkt, som man kan se i konstruktorn högst upp i filen, men vi återkommer till dessa senare. När *RunMenu* i klassen *Menu* körs, anropas ytterligare en metod; *PrintMenu* som skriver ut den synliga delen för användaren. I *RunMenu* finns även en switch som tar en igenom metoderna *AddNewCar*, *AddNewCandy*, *AddNewLace*, *AddNewTofu* samt *GetOrderList* beroende på knappvalet användaren gör. Det finns även ett case på knapptryck 0 i switchen och det är en return som avslutar programmet. Default caset i switchen anropas om användaren skriver in en siffra som inte är 0-5. I de senast nämnda metoderna beskriver namnet väldigt bra vad varje del gör. Jag tänker att jag kan beskriva *AddNewCar* samt *GetOrderLists* lite närmare. De fyra metoderna som lägger till en produkt i respektive lista är väldigt lika.

När man väljer ett knapptryck för att lägga till en produkt, exempelvis 1 som anropar metoden *AddNewCar* så rensas konsolen på text och man möts av ett nytt fönster med en förklaring vad man ska göra i kommande steg för att kunna "beställa" en bil. Ett nytt objekt av klassen *Car* skapas och användarens inmatade val sparas i objektets variabler och läggs sedan till i listan *carList* som redan har skapats i klassen *Menu's* konstruktor. En bil är tillagd i billistan och man kan gå tillbaka till menyn för att fortsätta sin beställning.

När man väljer knapptryck 5 för att skriva ut alla beställda produkter så anropas metoden *GetOrderLists.* Även här rensas konsolen på text och man möts av ett nytt fönster där alla ens beställningar listas. Metoden går igenom respektive produkts lista med en if-sats, är listan tom skrivs det ut att inga produkter av den varan är beställda. Annars skrivs alla produkter som finns i respektive lista ut med hjälp av en forloop.

Klassen *Menu* innehåller även två input och felhanteringsmetoder *ReadLineCheckInteger* och *ReadLineCheckString.* Dessa två är enbart för att hålla koden i varje metod lite renare från felhanteringskoden. *ReadLineCheckInteger* innehåller en try-catch i en while-loop som räddar programmet från att krasha om användaren skriver in bokstäver i stället för siffror. *ReadLineCheckString* är också en felhanteringskoll som egentligen bara kollar att användaren verkligen skriver något och inte lämnar ett tomt fält.

Egen analys och reflektion

Ska jag vara ärlig så tog det några vändor att nå nuvarande resultat. Jag har skapat två andra varianter av multifabriken innan jag kom fram till nuvarande lösning. Det svåra är att veta vad som egentligen är en korrekt lösning och inte då alla versioner har fungerat. Men i slutändan är jag nöjd över det här resultatet.

I den första versionen av multifabriken så byggdes menyn i klassen *Program* under metoden *Main* i stället för att ha en egen klass för menyn. Den största skillnaden om man jämför den första versionen med slutresultatet var att jag då skapade en abstract superklass som jag tänkte namnge något i stil med "*Order*", jag skapade sedan en lista av klassen *Order*. Alla produkters klasser sattes till subklasser av *Order*. Sedan när användaren gjorde sitt val skapades bara ett nytt objekt av vald produkt och lades in i orderlistan tillsammans med de andra produkterna oavsett produkttyp. Det här var en ganska smidig lösning av uppgiften. Men utskriften av alla beställningar blev inte riktigt lika lätt att påverka som slutversionen. Det hade förmodligen gått att grotta sig ner i en lösning för att få en lika bra utskrift på ett eller annat sätt även om man bygger på det här sättet. Men i uppgiftsbeskrivningen står det ju faktiskt att man bör ha en klass och lista för varje typ av produkt. Vilket också låter som en vettig idé i praktiken för att inte krångla till och blanda olika produkter. Därför ville jag fortsätta att bygga på ett annat sätt.

I den andra versionen av multifabriken var den största skillnaden konstruktorn i varje produkts klass. All kod som innehöll Console.WriteLine och alla variabler som tillsattes värden beroende på användarens input fanns i den här versionen i konstruktorn. Det enda man behövde i *Main* metoden för att lägga till en ny bil var att skapa ett nytt objekt av exempelvis klassen *Car* så kördes koden i konstruktorn och tog användaren till rätt funktionalitet. Därefter var det bara att lägga till den skapade bilen i rätt lista. Jag kommer ihåg att jag tyckte det här kändes som en bra lösning just då. Kodens uppbyggnad blev tydlig, man såg vad som gjorde vad. Mycket av koden som tillhörde exempelvis klassen *Car* hamnade i just den klassen. Frågan är bara hur mycket man egentligen vill bygga in i en klass då det känns som att man "låser" klassen lite mer när det är byggt på det sättet jag nu beskrev. Om man bygger på ett sätt med exempelvis metoder från en annan klass, exempelvis *Menu* i stället och har klassen *Car* ganska ren så kan man presentera sin bil på olika sätt med olika metoder och bilen fortsätter fortfarande att vara densamma. Det kanske låter otydligt när jag beskriver det så här, men det var de här tankarna som fick in mig på slutresultatets version.

Sista och nuvarande versionen. Efter mycket grubblande och många funderingar så kom jag i hamn med det här resultatet, som jag beskriver lite närmare högst upp i den här reflektionen under "kodbeskrivning". Här valde jag som sagt att skapa en helt fristående klass för menyn som jag sen anropar via metoden *Main* i klassen *Program*. Jag är nöjd med koden jag presenterar och känner att strukturen är tydlig. Den byggs på tydliga klass- och metodnamn och den är uppbyggd på ett sätt så vem som helst ska kunna sätta sig in i koden på ett enkelt sätt. Jag är även medveten om att det förmodligen finns mycket förbättringspotential i vissa delar av min kod. Ett exempel jag kan förbättra är felhanteringen av inmatningar. Man skulle kunna begränsa användaren mer än vad som görs i programmet för att användaren inte ska kunna skriva fel saker, i brist på kunskap så valde jag att avvakta med att utveckla den delen för att efterforska djupare kunskaper i ämnet.

Nackdelen med min uppbyggnad är väl att man behöver ändra på fler ställen än ett om man vill lägga till en produkt. Men jag tänker att det behöver man förmodligen i de flesta fall, mer eller mindre, när man ska lägga till något i ett projekt, men tanken är väl att man ska strukturera koden så att det blir så enkelt som möjligt att göra just det här. I mitt program för att lägga till en ny produkt behöver man skapa en ny klass för produkten och lägga till alla nödvändigheter för det. Sen behöver

man göra det jag nämner härnäst under klassen *Menu*: Skapa en ny lista för produkten. Skapa en ny plats i switchen under metoden *RunMenu*. Lägga till så att användare ser den nya produkten under metoden *PrintMenu*. Skapa en ny metod "*AddNewProduct*" där man skapar ett nytt objekt och lägger till i vald lista. Samt lägga till en utskrift för produkten under *GetOrderLists*. Trots detta så känner jag att min uppbyggnad är tydlig och enkel att förstå sig på. Om inte annat så finns det utvecklingspotential att göra det ännu enklare att kunna lägga till en ny produkt.

Allt här ovanför beskriver mitt arbetsflöde och vilka olika vägar jag har tagit för att hamna i den kodstruktur där jag är just nu. Jag har helt enkelt testat mig fram för att sen känna efter vilken lösning som känns mest praktisk. Jag är dock fortfarande aningen osäker på om man bör fylla på med mer information i själva produktklasserna eller inte, exempelvis bygga fler metoder där i stället. Alltså om mina första versioner av multifabriken skulle vara bättre på ett eller annat sätt. Men det är förhoppningsvis sådant man lär sig med tiden ju längre vi kommer i utbildningen. Hur man strukturerar objektorienterad kod på ett så lämpligt sätt som möjligt!

Länk till repo:

https://github.com/campus-varnamo/multifabriken-jajo21