Niklas Björk, WU21

Reflektioner inlämningsuppgift: Multifabriken, 3 Yhp

Beskrivning av koden

Koden är skriven med språket C# som en konsolapplikation och körbar i .NET Core.
Strukturen i koden består av Program klassen samt fyra extra klasser baserade på substantiven Bil, Godis, Snöre och Tofu i kravspecifikationen. Varje extra klass har tre metoder vardera. Metoderna är baserade på verben ”beställa produkter” samt ”lista produkter”. Metoden för att lista produkter använder sig av ytterligare en metod, att lista klassernas (Car, Candy, Lace och Tofy) innehåll, d.v.s. värdena som sätts i konstruktorn.
Varje klass innehåller satser som deklarerar variabler för innehållet i varje klass, samt en konstruktor som sätter värdet på variablerna vid skapande av nytt objekt av klassen.
I programklassen skapas fyra listor för de olika klasserna (typer), där varje ny skapad produkt av varje typ sparas. I programklassen skapas också huvudmenyn där de olika valen kan göras, d.v.s. beställa, lista beställningar samt avsluta. Huvudmenyn körs i en while-loop som styrs av en bool. Så länge inte while-loopen bryts (val sex, som sätter boolen till true) återkommer huvudmenyn efter varje val.
De olika valen styrs med en switch-sats. Switch-satsen använder sig i sin tur av metoderna i de olika klasserna för att utföra de önskade tjänsterna.

Skäl till vald lösning

Att skapa klasser för varje produkt var ett självklart val. Dels för att göra koden mer lättläst, men också för att på ett smidigt sätt kunna utveckla koden med fler produkter. Att lägga metoderna i varje klass blev också naturligt, då de anpassades till varje klass (produkt). Ligger de i varje klass, är det också lätt att förstå vart de hör hemma och gör koden tydligare och lättläst, både i produktklasserna och programklassen.
Varje klass har en konstruktor som säkerställer att varje nytt skapat objekt får de värden det ska ha när det skapas. Satserna i klassen som deklarerar variablerna är också ”protected”, vilket gör att värdena inte kan ändras utanför den egna klassen.
I programklassen deklareras listorna för de olika typerna (class). Jag valde att göra en lista för varje typ, då det i detta fallet blev både tydligare kod samt enklare användning. Varje lista innehåller bara en typ, vilket gör att när listan ska ”skrivas ut” i konsolen kan jag med en loop numrera samt inleda utskriften med specifik text.
För menyn valde jag att använda en switch-sats för att tydliggöra att det är olika val. Andra satser skulle kunna använts, som t.ex. ”if” men då hade det blivit många av ”else if”, och det ville jag undvika. Dessutom anser jag att det också blir lite otydligare kod med ”if-sats”.
I varje ”case” i switch-satsen anropas metoder i klasser och parametern för listan till specifik klass, skickas med i anropet.

Personlig reflektion av vald lösning samt andra alternativ

När jag började skissa på en programstruktur övervägde jag att använda mig av en superklass med namn som: class Products. Eftersom att alla produkter inte hade lika egenskaper, så såg jag ingen större mening med det. Skillnad hade det varit om varje produkt skulle ha lika egenskaper som; EAN-kod, produktnummer, leverantör, pris m.fl. plus de unika egenskaperna, då hade en superklass varit mer motiverat.

Skulle jag ändå gjort en superklass så medförde det, för mig, mer invecklad kod. Dels så skulle alla typer (class) hamna i samma produktlista. I och för sig kan det vara en fördel med bara en lista (om utbudet skulle vara större), men åter igen, så var det inte fallet denna gång. När sedan produkterna skulle listas, så skulle den samlade beställningslistan behöva sorteras, numrera beställda produkter, innehålla mer information för att inleda produktgrupper (type) vid utskrift i konsolen.

Just metoden, att lista produkterna skulle kunna bli något enklare, eller i alla fall mindre kod i programmet, då metoden bara skulle hamna i superklassen och ärvas av de andra klasserna. Dock blir den metoden något mer invecklad, då den behöver sortera typer, numrera produkter och även skriva ut om inget är beställt av en viss produkt. I denna kravspecifikation så finns inte sortering, numrering eller inledande text för varje produkt med, men om vi ser till UX, så blir det väldigt rörigt med en listning utan någon sortering eller indelning.

I övrigt så skulle jag kunna använt ”if-sats” istället för ”switch-sats” som jag skrev ovan, men switch visade tydligare funktionen.

Att köra menyn i en while-loop har jag inget bra alternativ till förutom att göra om den till en do-loop, men det blir i princip samma sak.

Efter vår handledning så förstår jag argumentet att hålla alla klasser rena från Console...(), för att klasserna inte ska behöva ändras om programmet inte körs som en konsol. När jag började koda uppgiften, jobbade jag med all inmatning- och utmatningskod {Console…} direkt i switch-satsen, men flyttade in dem i klasserna för att göra koden mer lättläst. Eftersom det i kravspecifikationen står att det ska vara en konsolapplikation så ändrar jag inte tillbaka denna gången, eftersom jag också tycker koden som den är nu blir mer lättläst och tydligare indelad.

Att ha med felhantering och undantag är väl troligt en självklarhet i programmering. I detta fallet tror jag dock inte det var syftet med uppgiften. Eftersom jag har haft gott om tid har jag ändå gjort en del felhantering. Önskar gärna feedback även på det, vad som var bra och vad som kan förbättras.

Programstrukturen som jag startade med att skissa upp ligger med som den sista sidan i denna pdf.

Multifabriken

class Program

- dynamiska Listor
  - Lista med beställda bilar
  - Lista med beställt godis
  - Lista med beställda snären
  - Lista med beställd tofu
- Meny (switch) : körs i en while-loop
  - Introduktion
  - Beställ bil (metod i class Car)
  - Beställ godis
  - Beställ snören
  - Beställ tofu
  - Lista din beställning
  - Avsluta programmet
  - Anropar metoden i classerna för input av värden.
  - Anropar metoden i classerna som skriver ut alla fält.
  - Avslutar while-loopen

class Car

- fält
  - string Registreingsnummer
  - string Färg
  - string Bilmärke
- metoder
  - Skriv ut alla fält: writePropertyForCar()
  - Skriva ut listan med beställningar: printListCar()
  - Inmatning av värden för new Car i lista för Cars: inputPropertyForCar()

class Candy

- fält
  - string Smak
  - string Antal
- metoder
  - Skriv ut alla fält: writePropertyForCandy()
  - Skriva ut listan med beställningar: printListCandy()
  - Inmatning av värden för new Candy i lista för Candy: inputPropertyForCandy()

class Lace

- fält
  - string Färg
  - string Längd
- metoder
  - Skriv ut alla fält: writePropertyForLace()
  - Skriva ut listan med beställningar: printListLace()
  - Inmatning av värden för new Lace i lista för Lace: inputPropertyForLace()

class Tofu

- fält
  - string Volym
  - string Kryddning
- metoder
  - Skriv ut alla fält: writePropertyForTofu()
  - Skriva ut listan med beställningar: printListTofu()
  - Inmatning av värden för new Tofu i lista för Tofu: inputPropertyForTofu()