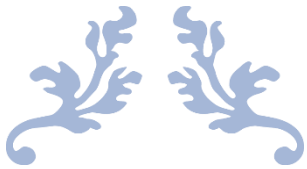

MULTIFABRIKEN AB

Inlämningsuppgift - Programmering med C# och objektorienterad design

DEN 13 NOVEMBER 2020

WIVIANNE GRAPENHOLT
grwi20ql@student.ju.se

Innehåll

Funktioner och Struktur

När programmet startar, i Program.Main, anropas funktionen Order.CreateOrder som skapar och returnerar en ny lista, order.

Ett Välkomstmeddelande skrives ut och därefter inleds en do-while loop som styrs av en bool variabel, main. I loopen anropas funktionen Menu.ShowMainMenu som presenterar en meny för användaren. Kundens val (1, 2 eller 0) fångas upp och avgör vidare flöde i programmet med hjälp av en if-else-sats.

Huvudmeny: val 1

Om användaren väljer att beställa rensas konsolen, sedan anropas ShowSubMenu med order som argument. Hela undermenyn ligger i en do-while loop som styrs av den lokala bool variabeln sub. En undermeny skrivas ut med de olika produkterna samt möjligheten att gå tillbaka till huvudmenyn.

Produktmeny

Inmatat värde fångas upp och modifieras av ToUpper. Valet styrs med en switch-sats som avgör vilken typ av produkt och därmed vilken metod, orderCar/Sweets/Lace/Tofu, som anropas men flödet är i stort sett det samma.

I fall valet är B, G, S eller T

Skapas en lokal variabel som anropar *Typ*.order*Typ*. Här töms skärmen på nytt och detaljer om produkten efterfrågas. Dessa fångas upp och med hjälp av inmatade värde anropas vald typs konstruktor för att generera ett objekt som sedan returneras.

I fall valet är S, T eller G

I tre fall, Lace, Tofu och Sweets, konverteras inmatat värde till siffror, double eller int.
För att fånga fel inmatningar utan att programmet kraschar används metoden ReadDouble/ReadInt som med hjälp av TryParse som antingen returnerar en korrekt variabel eller ett felmeddelande som låter kunden försöka igen.
När inmatningen är korrekt skickas listan och det nya objektet som argument till Order.AddToOrder. Nu läggs objektet till i listan order med funktionen Add och kunden får en bekräftelse att beställningen gick igenom via metoden ShowConfirmation: återigen töms konsolen innan meddelandet skrivs ut och utskriften hålls kvar tills användaren trycker på valfri tangent.

I fall valet är H

Skärmen töms, sub sätts till false, iterationen stoppas och flödet återgår till huvudmenyn.

I fall inget annat val matchar (default)

Skärmen töms, ett felmeddelande visas, sub sätts till true, en ny iteration av undermenyn startar.

Huvudmeny val: 2

Om kunden istället väljer att visa order töms först konsolen sedan kontrolleras att listan order inte är tom med en if else-sats. Då anropas Order.ShowOrder med order som argument. Här itereras listan i en foreach loop och funktionen Product.ShowProducts anropas för varje produkt. Metoden skriver helt enkelt ut informationen som är anpassat till instansens attribut. Även här hålls informationen kvar, väntar på tangenttryck innan skärmen rensas.

Huvudmeny val: 0

Konsolen rensas, ett avskedsmeddelande visas och programmet avslutas med nyckelordet return.

Klasser

Jag har valt att utöver Program och System implementera följande klasser:

Order

Är en abstrakt, superklass till Menu och har endast statiska publika metoder som klassmedlemmar: CreateOrder, AddToOrder och ShowOrder. Här genereras, modifieras och exponeras beställningar av produkter efter kundens önskemål.

Menu

Är också en abstrakt klass har också bara statiska metoder som klassmedlemmar, två publika: ShowMainMenu, ShowSubMenu och en privat ShowConfirmation. Här hanteras utdata i form av menyerna och även indata, menyval, i produktmenyn.

Product

En abstrakt parent-class till Car, Sweets, Lace och Tofu. Förutom två skyddade properties, en string och en double, innehåller klassen endast en publik abstrakt metod för att visa beställda produkter, ShowProducts.

Car, Sweets, Lace och Tofu

Har alla varsin publik metod som skriver över den ärvda metoden, ShowProducts, och är anpassat för att presentera den vald produkten för användaren.
Den publika, statiska metoden orderCar, orderSweets, orderLace och orderTofu hanterar ut- och in data vid beställning samt genererar och returnerar en ny instans. På så sätt kan samtliga konstruktorer vara privata.
Car som är den enda sub klassen som har egna properties utöver de som ärvs av Product. Konstruktorn kräver därför tre string argument. Double propertyn som ärvs sätts till 1.
Sweets, Lace och Tofu ärver sina properties rakt av från Product därför har de varsin konstruktor som behöver två argument. Här ligger även de privata statiska metoderna, ReadInt och ReadDouble, för att konvertera inmatade värden till int respektive double.

Analys och Reflektion

Överskådligt

Jag har valt att hålla Program klassen till endast huvudmenyvalen för att man ska få en snabb uppfattning av vad programmet gör i stora drag. Med tydliga självförklarande och statiska metodanrop, som guidar vidare till i vilken klass programflödet fortsätter.
Jag har haft en tanke om att även flytta hela huvudmenyn till Menu klassen men då hade Program bara innehållit introduktionsmeddelandet och ett enda metodanrop. Vilket jag anser hade påverkat överblicken av flödet negativt.
Ursprungligen existerade inte klasserna Order och Menu, de metoderna låg i Product eller Program. Något som jag upplevde det som rörigt och ologiskt.
Jag gjorde Order till parent-class över Menu var mest för att jag tänker mig att Order utför kundstyrda processer i ”back-end” medan Menu samlar in och presenterar informationen till användaren.
Namngivningen av klasser, metoder och variabler anser jag hjälper till att göra programmet överskådligt.
Att i produktmenyn modifiera kundens inmatningar, till versaler. Anser jag ökar läsbarheten av koden eftersom min lösning gör att koden hålls enklare. Alternativet hade varit att lösa det med att ange villkor för både gemener och versaler vilket hade tvingat mig att antingen ha flera case i switch-satsen eller att använda if-else-satser istället.

Vidareutveckling och förbättringar

En effekt av att jag har metoderna för att efterfråga och samla in attribut-värdena i varje subklass av Product blev att jag för enkelhetens skull inte skulle behöva först returnera dessa för att sedan användas för att anropa en konstruktor. Vilket i sin tur ledde till att jag valde lösningen att sätta konstruktorerna till privata. Som jag ser det möjliggör det även ett smidigare sätt att kunna plocka bort och lägga till produkter i fabriken om man så önskar.
Jag har valt att spara objektens attribut ”amount” som double del för att två av typerna passade bättre som decimaltal i de måttenheterna, och det är enkelt att konvertera heltal till decimal än vice versa. Dels för att kunna lägga till pris per styck, meter och liter och på ett enkelt sätt implementera funktioner för att räkna ut och skriva ut pris.
Både för enskild produkt samt totalsumman för kundens order. Av samma anledning har jag hårdkodat denna variabel för instanser av Car, man kan bara beställa en bil åt gången vilket känns rimligt.
Metod för att räkna ut pris borde lämpligen placeras i Order och funktion för att visa resultatet anser jag hör bäst hemma i Menu. Vilket gör att valet med att Menu ärver från Order känns ännu mera motiverat.
En förbättring av strukturen som kan underlätta för att vidare utveckla programmet är att låta Product ha virtuella metoder för alla de olika metoderna i sub-klasserna, som vid behov skrivs över.

Användarvänlighet

För mig är även struktur och tydlighet för användaren viktig.
Därför har jag varit frikostig med att konsolen töms på innehåll mellan olika funktioner och att informationen finns kvar tills kunden hunnit läsa klart.
Att bekräfta order genom ett meddelande och referera till menyval 2 bidrar till tydligheten.
Jag använder ToUpper i produktmenyn, fångar upp felinmatningar och att låta kunden göra nya försök tills det blir rätt. Allt för att öka användarvänligheten.