Labbrapport: Labb 4

Författare: Minna Rabhi Hallner, Ida Johansson

Datum: 2025-12-15

Kursnamn: GIK299 – Objektorienterad Programmering

Examinator: Elin Ekman och Ulrika Artursson Wissa

Innehåll

1. Introduktion

Labben är programmerad i programmeringsspråket C# och använder objektorienterade principer. Syftet med labben var att utveckla våra kunskaper kring hur klasser och andra typer interagerar med varandra. VI har även arbetat med hur en kan hantera felaktiga indata från användaren.

Själva uppgiften gick ut på att skapa ett konsol-program där användaren kan utifrån en meny välja att mata in data om en person och lägga till denna i en lista, eller skriva ut information om alla inlagda personer.

I denna labbrapport beskriver vi hur vi gått till väga, vilka hinder vi stötte på längs vägen samt reflekterar över de lärdomar vi tagit med oss efter den slutförda rapporten.

2. Metod

2.1. Verktyg

I labben har följande verktyg varit delaktiga för att utföra uppgifterna:

- För att bygga samt testa vår applikation användes Visual studio 2022, version 17.14.19 samt ramverket .NET 8.0.
- NuGet Package Manager 6.14.1
- Visual Studio IntelliCode och Visual Basic Tools för att använda programmet och underlätta skrivning av kod.
- Föreläsningar av Robert Pilstål samt lektioner och programmeringsstugor under handledning av Elin Ekman och Ulrika Artursson Wissa.
- Google Gemini för att förtydliga begrepp och koncept.
- GitHub Copilot för att förstå fel i error-listan i Visual Studio 2022.
- W3Schools, GeeksForGeeks och StackOverflow för att ge kodexempel och förslag på tillvägagångssätt.

2.2. Stegvis beskrivning av tillvägagångssätt

För att lösa sprint 1 av uppgiften använde vi oss av pseudokod i form av kommentarer för att få en överblick av vilka steg vi behövde ta. Vi skrev sedan koden under varje kommentar som hörde till steget. I ett första utkast av kod skapade vi upp alla delar som hör till persondata i en fil, som vi sedan strukturerade upp i olika filer: Person.cs, Hair.cs och Gender.cs. Utöver detta använde vi Program.cs för att instansiera objekt av Person-klassen och köra programmet. Vi bestämde ett tidsspann där en av oss var ”driver” och den andra ”navigator” för att se till att båda fick skriva kod.

För sprint 2 hade vi samma upplägg för planeringen. Vi skrev pseudokod på lämpliga ställen i koden från sprint 1. Vi utgick sedan från dessa för att skriva riktig kod som fyllde kriterierna från uppgiftens kravlista. Vi lade till metoderna AddPerson() och ListPersons() till Person-klassen. Inom AddPerson-metoden hade vi ett antal inputs för persondata med felhantering för att hantera om användaren matar in felaktiga data. ListPersons() använde sig av den tidigare överskrivna ToString-metoden. Arbetsfördelningen såg i stort sett likadan ut som för sprint 1.

2.3. Förutsättningar för att göra labben

För att kunna köra koden behöver IDE:n Visual Studio 2022 och ramverket .NET 8.0. Programmeringsspråket som används är C#.

2.4. Testning av koden

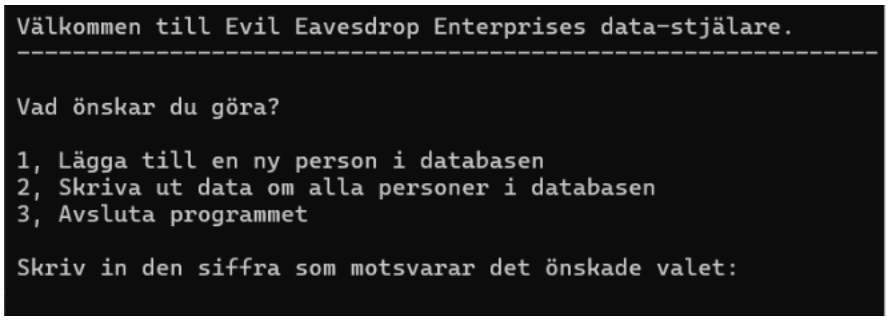
För att testa koden har vi använt oss av debuggern för att analysera hur koden fungerar i sin helhet. Vi har även använt oss av Error List i vår IDE som beskriver de fel eller varningar som koden innehåller. När vi hade implementerat felhantering för alla inputs körde vi programmet och matade medvetet in felaktiga data för att säkerställa att alla möjliga fel blev hanterade.

2.5. Etiska överväganden

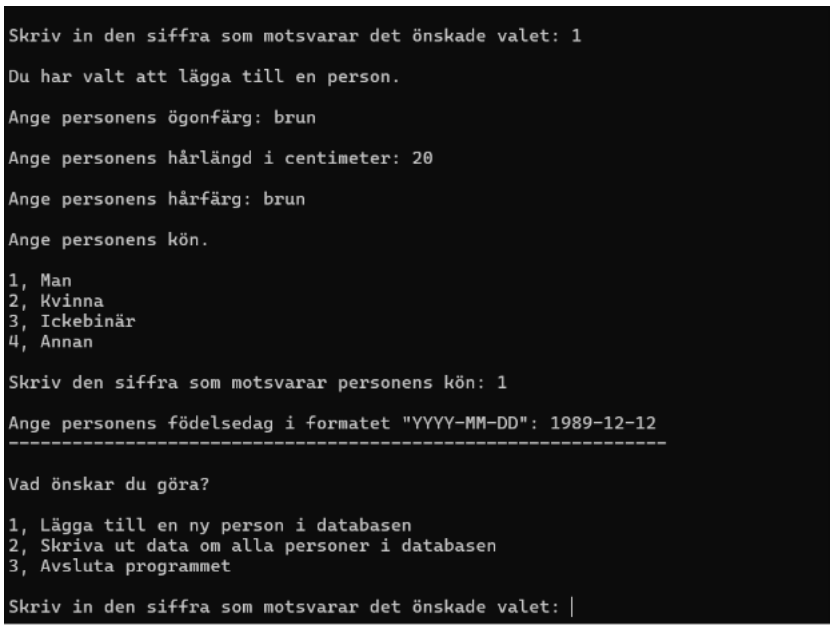
I och med att inga verkliga personer har använts för den här uppgiften har inga etiska överväganden varit nödvändiga.

3. Resultat

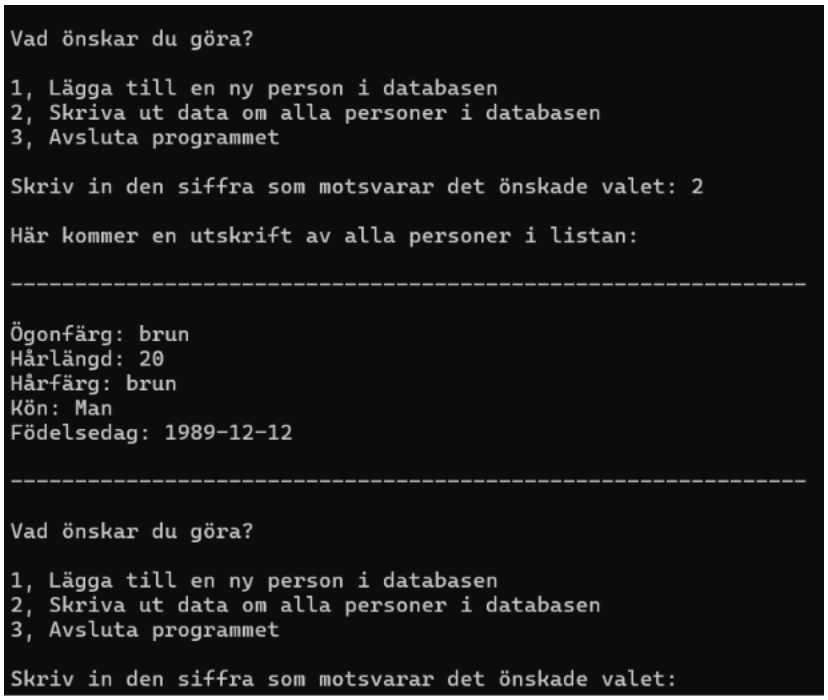
Efter att ha avslutat sprint 1 skriver programmet ut information om objektet “person1” där den personliga data har hårdkodats in i instansen.
Efter sprint 2 fungerade programmet som förväntat och efter ett så kallat ”duck-test” ansåg vi att koden uppfyllde kravlistan.



Figur 1: Startmenyn.

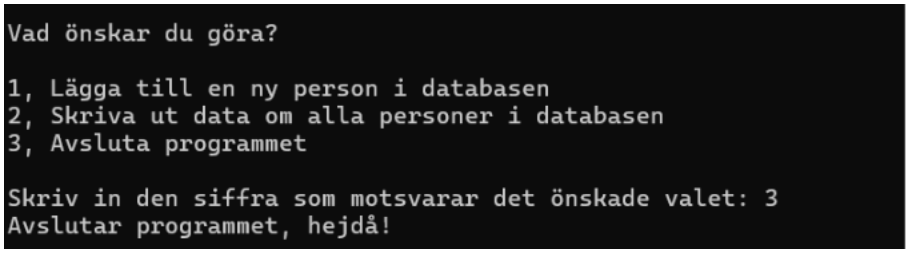


Figur 2: Korrekt inmatad data.



Figur 3: Utskrift av personen

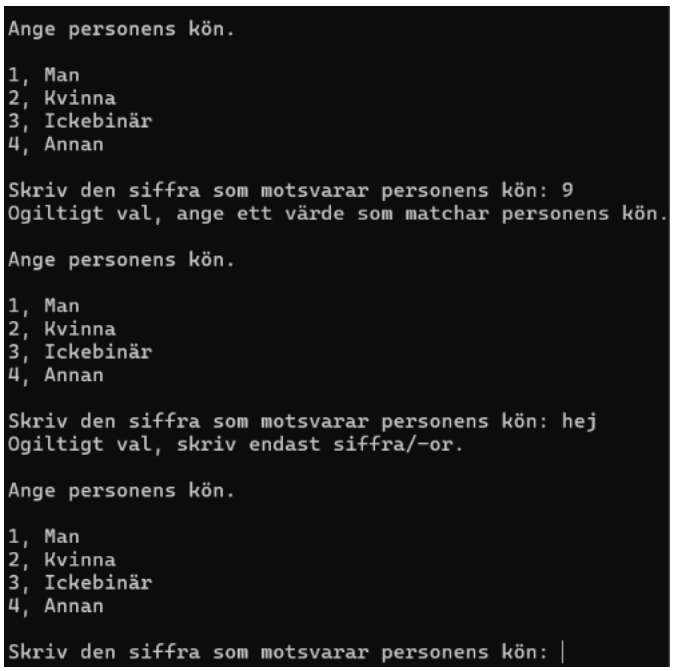
.



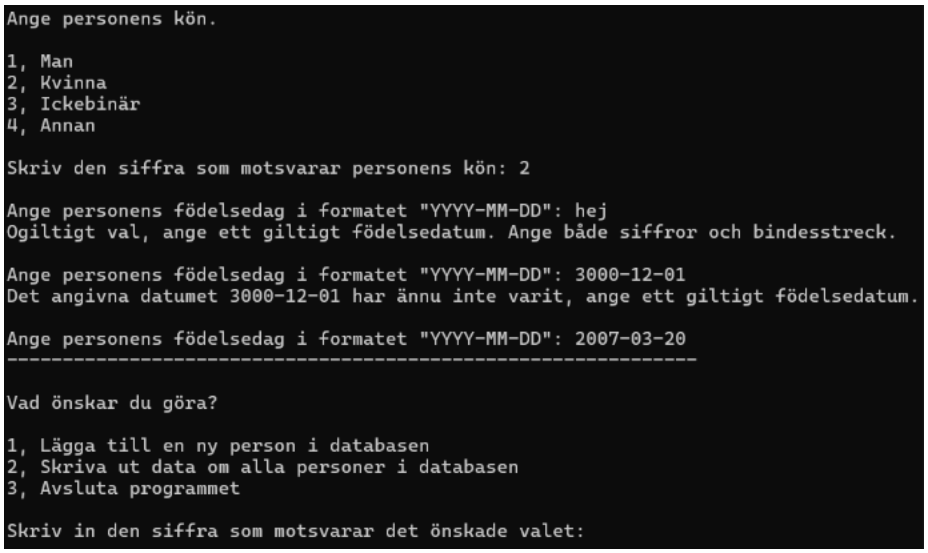
Figur 4: Avslutar programmet.

När vi testade felhanteringen av exempelvis hårlängden upptäckte vi att det fanns flera möjliga fel. Till exempel negativa tal eller att användaren matar in bokstäver eller andra tecken. För att hantera det använde vi oss av både en if-sats samt try och catch. Det tillät oss att skriva ut mer beskrivande felmeddelanden som förklarar för användaren vilket fel de gjort och vilken input som förväntas. Vi bäddade in if-satsen i try-blocket och även hela try-catch inom en do-while-loop för att säkerställa att användaren matar in korrekt data innan programmet går vidare.

Figur 5: Fel input i menyn.



Figur 6: Felaktig input för persondata.



Figur 7: Felaktig input för datum.

4. Diskussion och reflektion

Programkoden till denna labb skrevs utifrån en kravlista på funktionalitet. Koden resulterade i ett fungerande program, men labben kunde ha lösts på andra sätt med samma resultat.

Koden reviderades flera gånger under arbetes gång, allt eftersom vi lärde oss mer om C#. Vi stannade ofta upp och läste på om olika koncept och hur de kunde användas, för att sedan implementera våra nya kunskaper i koden.

4.1. Diskussion kring resultat

Vi valde att använda oss av DateOnly i stället för DateTime eftersom tiden var irrelevant för det här programmet. Det är ett bättre och enklare alternativ när endast datum ska hanteras (Microsoft, 2024). För att kunna jämföra det angivna födelsedatumet med dagens datum behövde vi instansiera en DateTime.Now och omvandla det till en ny DateOnly-variabel.

För felhantering använde vi oss av ett generellt catch-block. I framtiden skulle vi kunna använda throw för att instansiera unika exceptions och på så vis skriva ut beskrivande felmeddelanden för dessa specifika exceptions.

När vi skapade felhantering för input om färg valde vi att låta användaren skriva in vad som helst så länge det är längre än tre bokstäver. Anledningen till detta är att det finns otroligt många nyanser som man skulle kunna vilja beskriva någons hår eller ögon med. Vi har dock inte funnit någon färg som består av mindre än tre bokstäver och satte en gräns därefter.

```
Skriv in den siffra som motsvarar det önskade valet: 1

Du har valt att lägga till en person.

Ange personens ögonfärg: 40
Ogiltigt val, ange en färg med bokstäver.

Ange personens ögonfärg: bu
Ogiltigt val, ange en riktig färg.

Ange personens ögonfärg: blå

Ange personens hårlängd i centimeter: -5
Ogiltigt val, ange ett värde över 0 cm.

Ange personens hårlängd i centimeter: hej
Ogiltigt val, skriv endast siffra/-or.

Ange personens hårlängd i centimeter: 30

Ange personens hårfärg: bä
Ogiltigt val, ange en riktig färg.

Ange personens hårfärg: 50
Ogiltigt val, ange en färg med bokstäver.

Ange personens hårfärg: brun
```

Figur 8: Fel input för persondata, inklusive färg.

4.2. Reflektion kring sprint 1

Några av de erfarenheter vi tagit med oss från sprint 1 är hur man använder enums, structs och hur man överskriver virtuella metoder så som ToString(). Även vad konstruktorns funktion i programmets helhet är och hur man kan kombinera det med olika datatyper.

4.3. Reflektion kring sprint 2

Under sprint 2 blev vi mer bekväma i arbetssättet “driver-navigator” och det blev naturligt att byta mellan dessa positioner oftare. Om vi kände att en suttit på “driver-sidan”, så såg vi till att jämna ut detta vid nästa arbetstillfälle.

Vid skapandet av AddPerson-metoden stötte vi på flera koncept som var nya för oss. Exempelvis hur en hanterar felaktig eller korrekt angivna data på lämpligt sätt. I slutet av arbetet var det tydligt att vi fått en bredare kunskap i programmeringsspråket genom programmeringsstugor, föreläsningar samt lektioner eftersom vi kunde gå tillbaka och förenkla den kod vi tidigare skrivit. Exempelvis hade vi till en början en mycket komplicerad felhantering där varje input hade en egen, eller till och med två bool-variabler. När vi sedan istället såg till att använda en och samma bool-variabel för samtliga delar inom metoden blev koden mycket mer lättläslig.

I den första sprinten fick vi ytterligare kunskap kring hur en använder konstruktorn. Det underlättade arbetet i sprint 2 när ett nytt objekt av Person-klassen i AddPerson-metoden skulle instansieras. ToString-metoden och kunskaperna om hur den fungerar kom väl till användning även i sprint 2. Vi använde den för att göra en komplett utskrift i ListPersons-metoden.

4.4. Reflektion kring alternativa lösningar

Ett alternativ till lösningen för datumhantering hade varit att använda bara DateTime, även för användarens input. För att sedan skriva ut detta hade vi kunnat använda ToString-metoden för att formatera utskriften till att bara innehålla datum. Men eftersom Microsoft (2024) beskrev DateOnly-strukturen som en smidigare lösning valde vi det.

Vi valde att lägga all felhantering i AddPerson-metoden runt varje input. Ett alternativ till detta hade kunnat vara att placera en del av felhanteringen i set-metoden till de properties som lagrar data om personen för att säkerställa att rätt slags data inmatas. I och med att vi inte använde de properties någon annanstans ansåg vi att det räckte med felhantering i metoden.

Frågor till AI-verktyg

Verktyg: Gemini

Fråga/prompt: ”Om man har en struct i en annan klass än program, hur anropar man den? C#”

På vilket sätt svaret användes: Vi använde Geminis funktion ”Guidad inlärning” för att inte få ett svar direkt, utan Gemeni vägleder oss istället framåt med hjälp av frågor vi får besvara innan den ger oss mer information. På detta sätt är vi säkra på att vi förstår den information vi fått, innan vi går vidare till nästa steg. Svaret vi fick hjälpte oss att förstå hur man kan anropa en struct både inom och utanför en klass, genom instansiering. Den förklarade även hur man kan tilldela värden till de olika värdena i structen och hur man kan anropa ett specifikt värde i structen och göra en utskrift med det.

Referenser

Microsoft (14 december 2024). Så här använder du strukturerna DateOnly och TimeOnly. Microsoft Learn. https://learn.microsoft.com/sv-se/dotnet/standard/datetime/how-to-use-dateonly-timeonly