Labbrapport: Labb 4

Författare: Tyko Törngren Ljunggren

Datum: 2026-2-13

Kursnamn: GIK299 – Objektorienterad Programmering

Examinator: Elin Ekman och Ulrika Artursson Wissa

Innehåll

1. Introduktion

I denna labb har jag använt mig av objektorienterad programmering och objektorienterade principer för att skapa ett dynamiskt program som tar emot, hanterar och presenterar data för användaren. Labbrapporten går igenom hur arbetet genomfördes, de redskap och program som användes samt reflektioner om proccessen, vad som gick bra, vad som inte gick bra och vad som skulle kunna göras annorlunda

2. Metod

2.1. Verktyg

Jag har använt mig av JetRiders IDE, Rider, för att skapa och testa koden, ChatGPT samt Windows C# dokumentation för att bättre förstå koncept och som nedan nämnt ChatGPT för att skapa en bit av koden.

2.2. Stegvis beskrivning av tillvägagångssätt

Den första sprinten var ganska självklar då de givna kraven var tydligt och specifikt givna, jag följde punktlistan på krav uppifrån och ner tills alla krav var uppnådda. Eftersom jag gjorde denna laboration själv var min arbetsstruktur mer lös, jag avsatte tid åt att jobba med uppgiften, satte tydliga mål för vad jag hoppades uppnå på den tiden, och sedan satte jag mig ner och försökte uppnå de målen på den tiden, ibland hann jag, ibland inte. Inför den andra sprinten var mitt arbetssätt ganska lika, jag följde listan av krav, men arbetsuppgifterna var mer öppna och vid flertalet tillfällen fick jag ta beslut om hur en uppgift skulle lösas, och ofta förändrades valen allt eftersom för att effektivisera och bättre matcha kraven.

2.3. Förutsättningar för att göra labben

Jag använde mig utav Rider som IDE för att skapa och testa programmet, vilken var min valda IDE för de tidigare labbarna, i Rider kan man lätt skapa c# program med .Net upp till 9.0. Jag behövde inte ändra på några ytterligare inställningar för att få programmet att funka.

2.4. Testning av koden

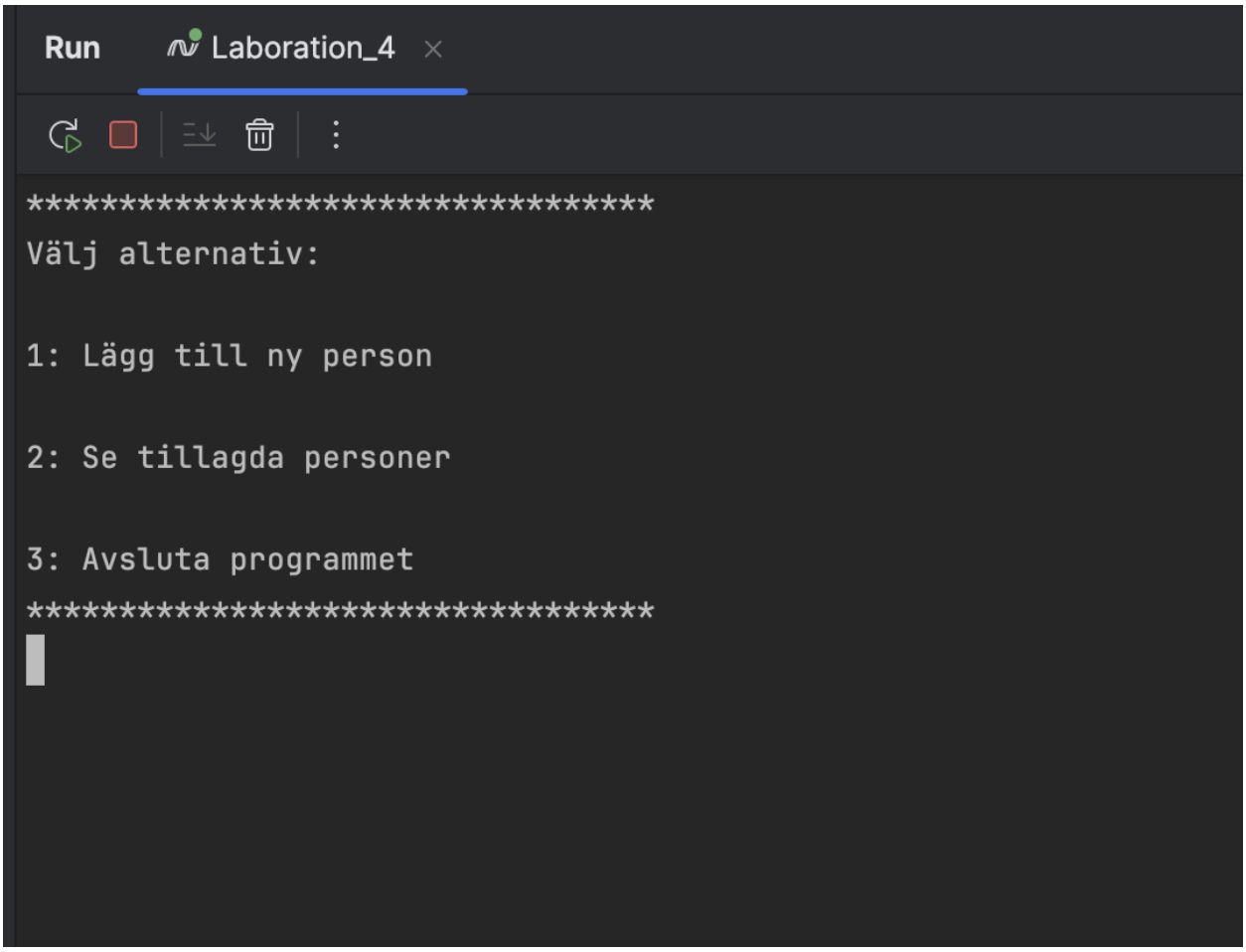
Koden testades på två huvudsakliga sätt, normalt och elakartat. Normala tester gjordes genom att sätta ett tydligt mål eller en uppgift, så som att lägga till en person och se vad som händer. Funkar programmet som tänkt? Uppstår problem eller oklarheter när man kör programmet som de ska? Elakartade tester sker genom att vid varje steg där det går, göra allt man kan för att få fel att uppstå, som att tex mata in fel sorts data. Detta kunde ske strukturerat, som ovan att ett givet mål/uppgift ges, eller mera “trial and error” där man kör igenom programmet utan plan och bara försöker få någonting att gå fel. Jag upptäckte en del fel igenom processen, speciellt vid de tillfällen då jag valde att ändra på funktioner i programmet och andra delar av koden inte hängde med.

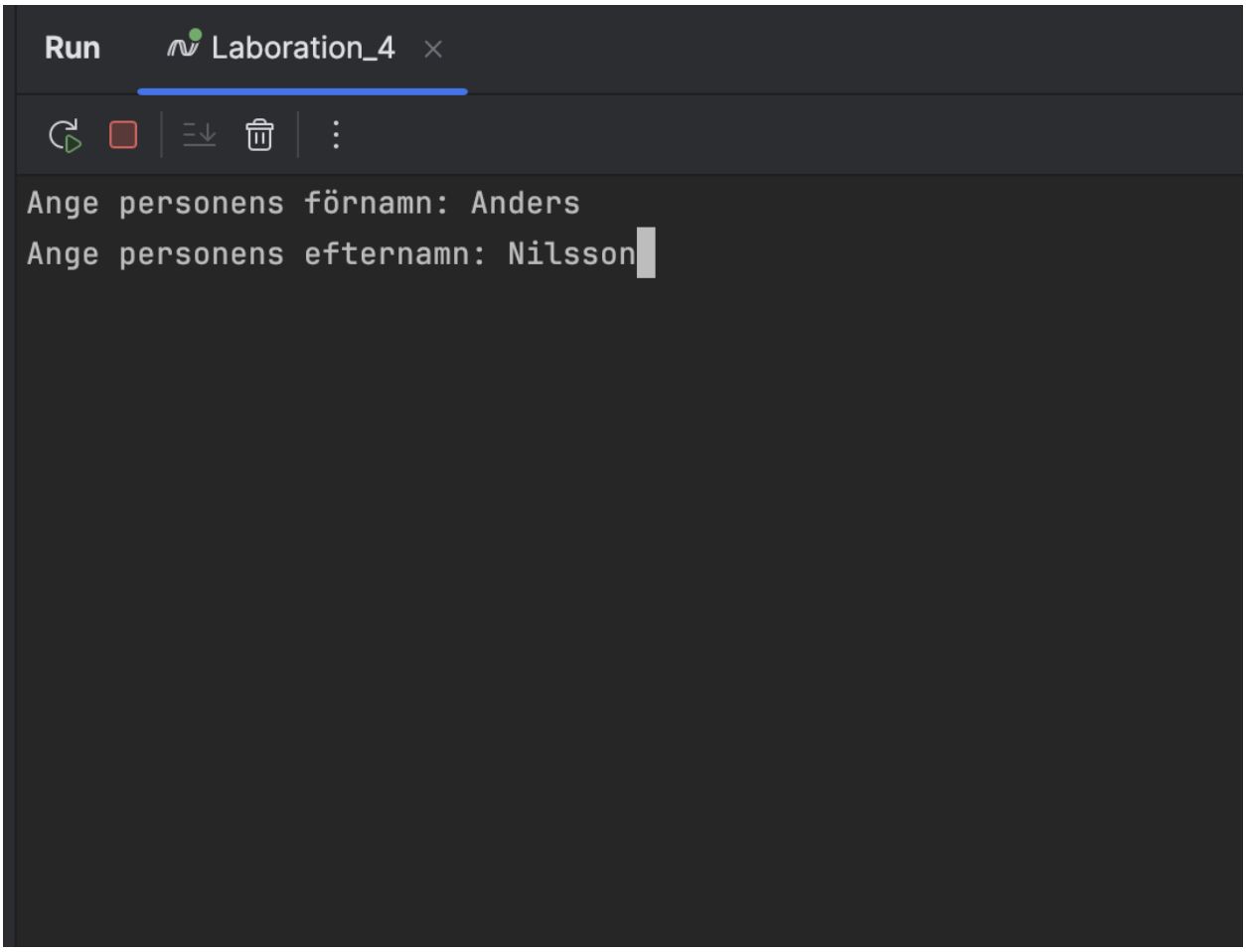
2.5. Etiska överväganden

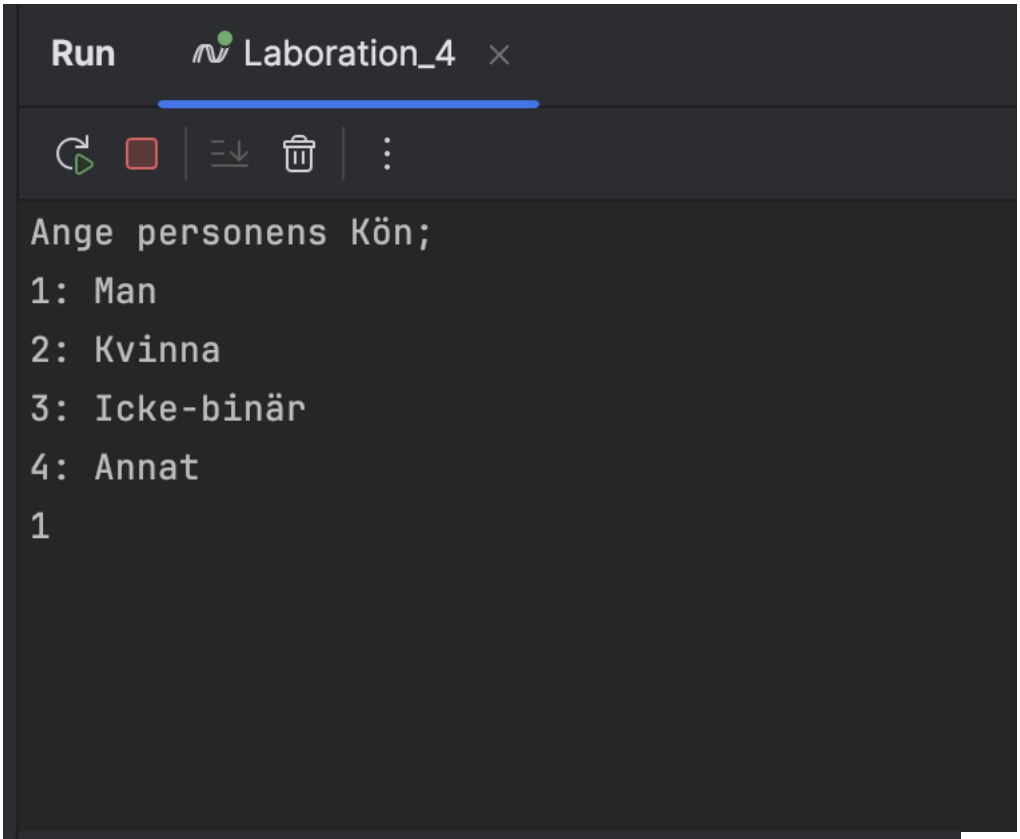
Eftersom datan som samlades in inte tillhörde någon riktig person, och kanske viktigare, inte sparades i någon databas för senare användning känner jag att de etiska överväganden var små. Dessutom var den personliga datan inte särskilt känslig, födelsedag, kön och hår färg/längd. även om det är personligt, kan det troligtvis inte användas för att åstadkomma några större skador.

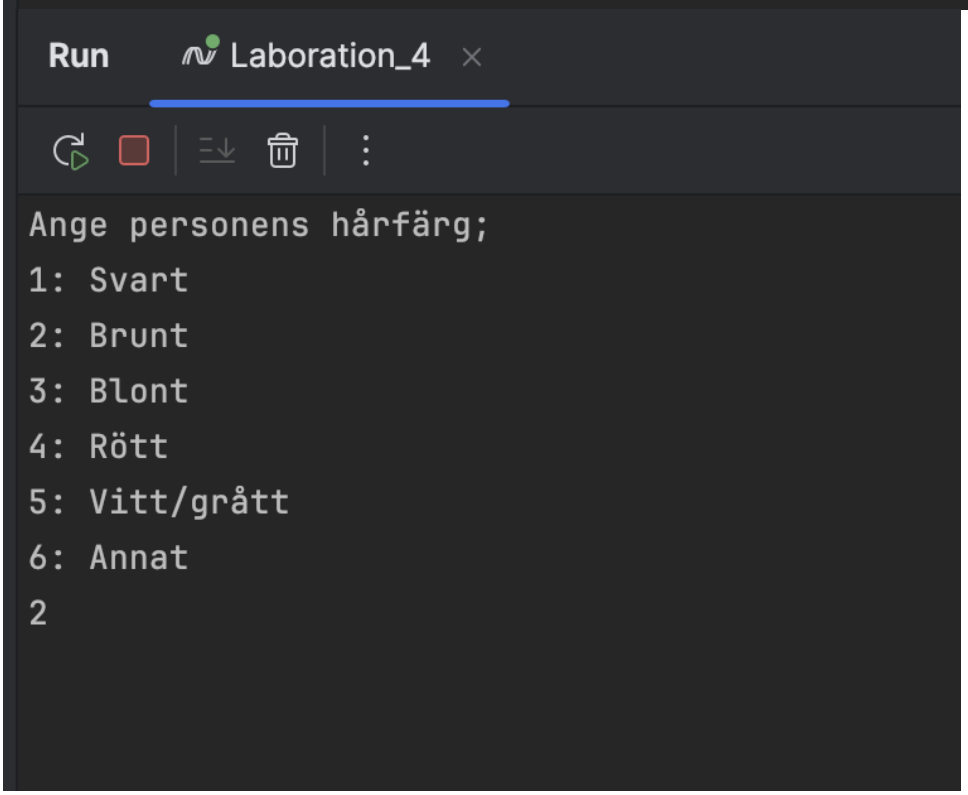
3. Resultat

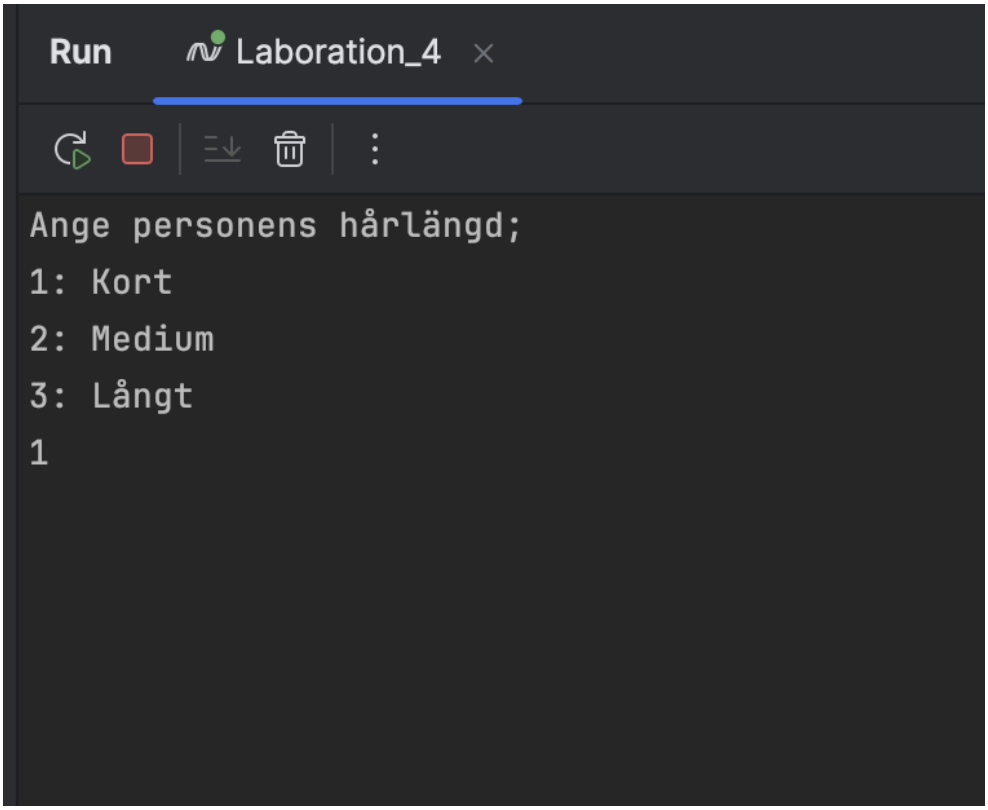
Programmet fungerar enligt kraven, man kan lägga till och visa personer på en skapad lista innehållande “Person” objekt, Person-klassen har all relevant information och skapade objekt läggs i en Personlista för förvaring. Jag tog inte några bilder under processen vilket innebar att jag inte har några bilder på fel och tidigare kod. De tidigaste varianterna av valideringen var bara string-iputs och valideringen skulle ske efteråt. De flesta felen uppstod efter att valideringsprocessen ändrats och den resterande koden inte hängde med.

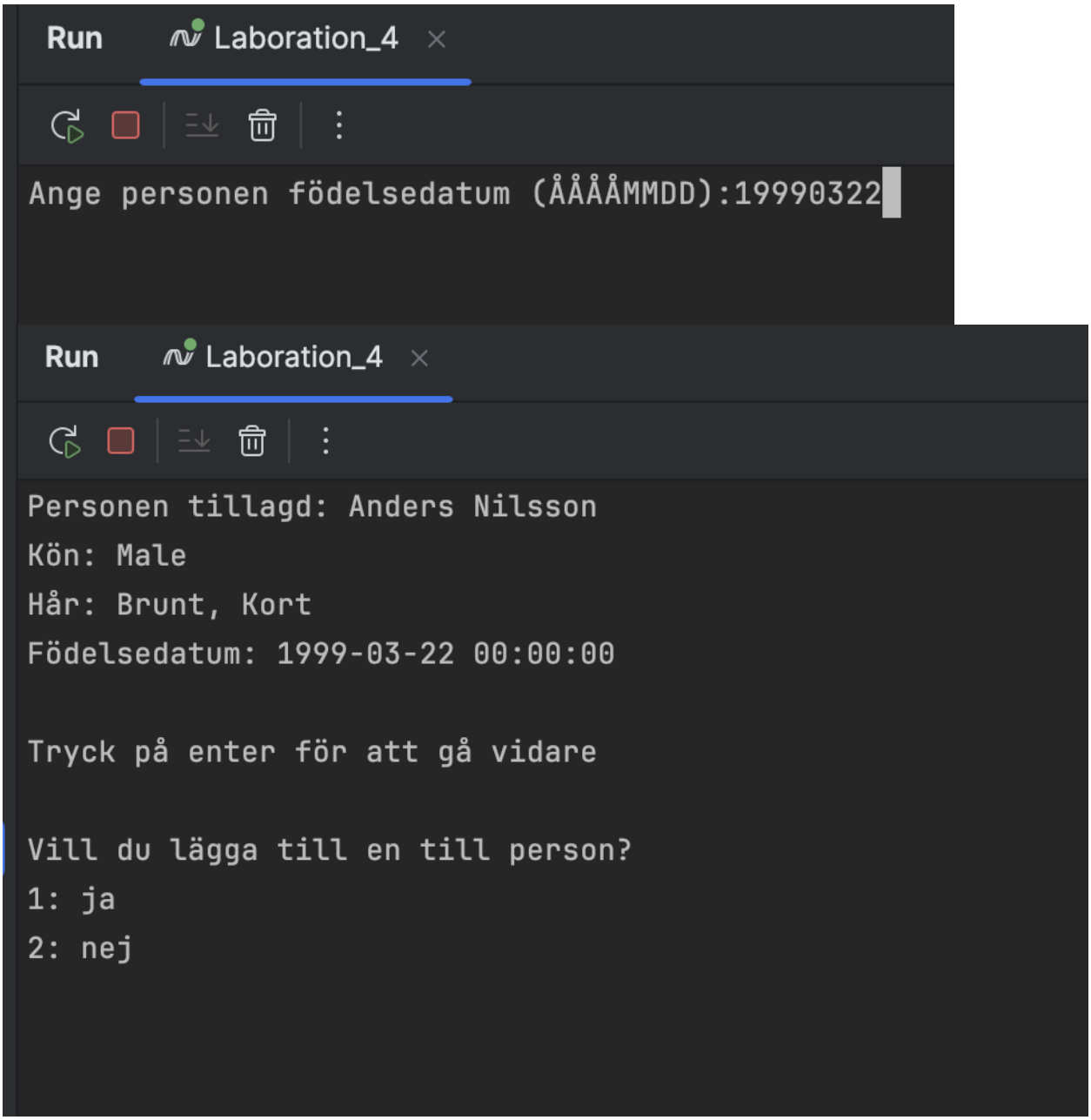

Run
Laboration_4
****
Välj alternativ:
1: Lägg till ny person
2: Se tillagda personer
3: Avsluta programmet
****

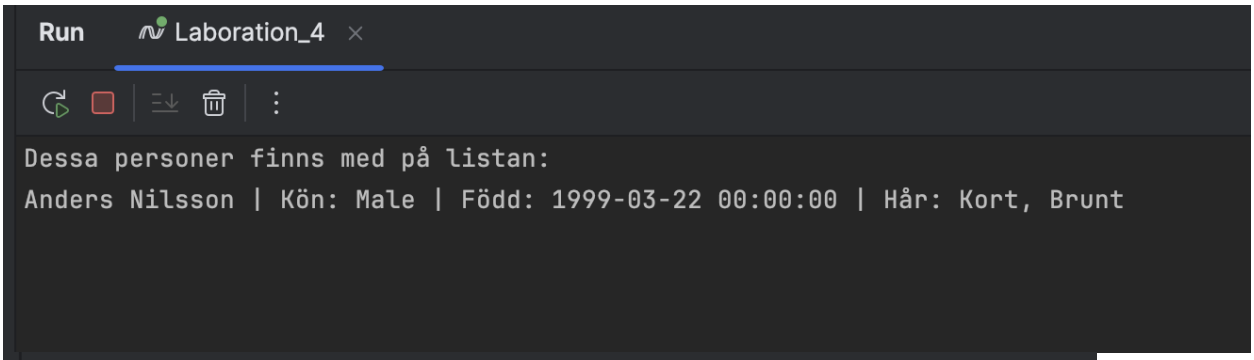

Run
Laboration_4
Ange personens förnamn: Anders
Ange personens efternamn: Nilsson

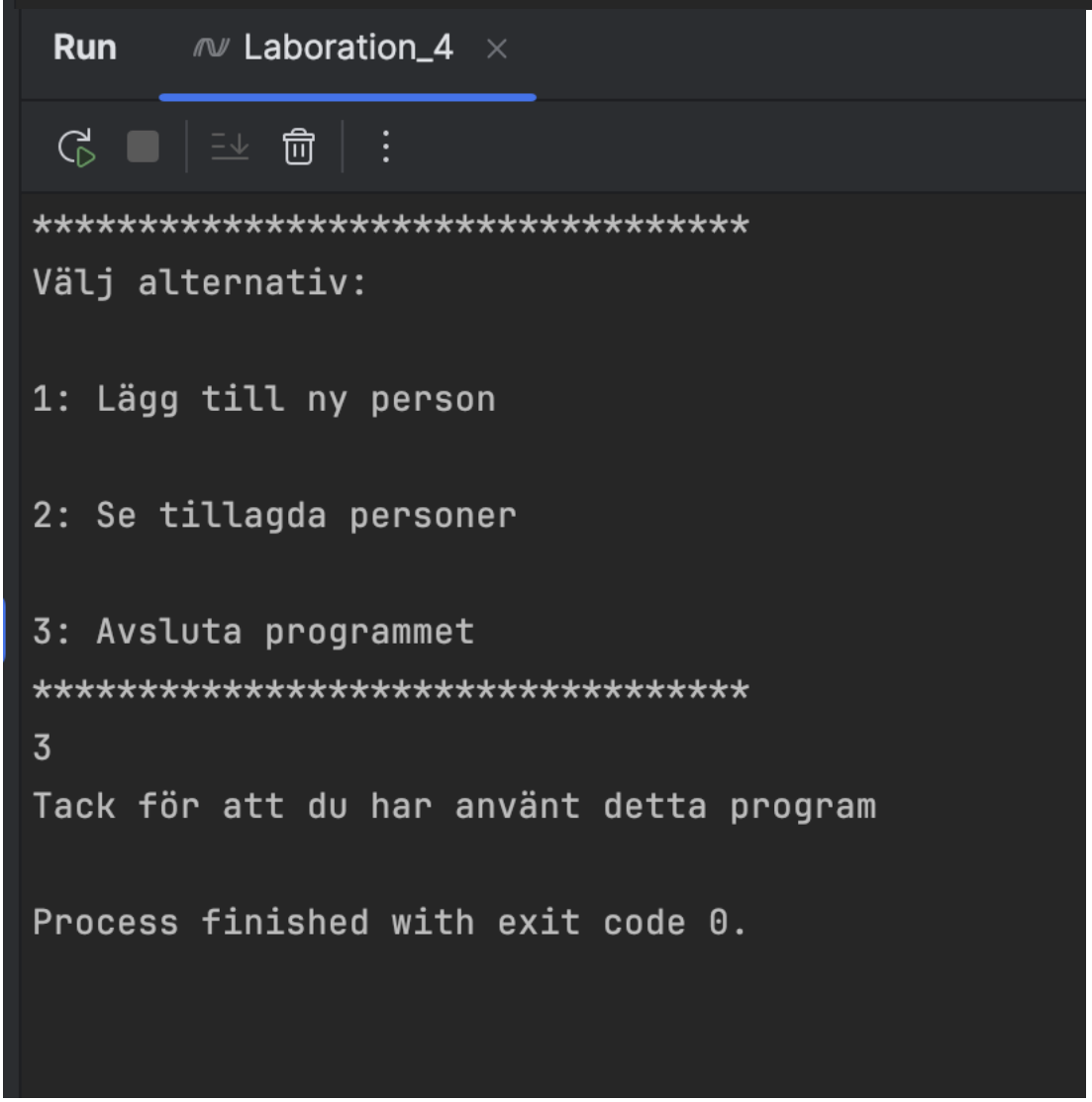

Run
Laboration_4
Ange personens Kön;
1: Man
2: Kvinna
3: Icke-binär
4: Annat
1


Run
Laboration_4
Ange personens hårfärg;
1: Svart
2: Brunt
3: Blont
4: Rött
5: Vitt/grått
6: Annat
2


Run
Laboration_4
Ange personens hårlängd;
1: Kort
2: Medium
3: Långt
1


Run
Laboration_4
Ange personen födelsedatum (ÅÅÅÅMMDD):19990322
Run
Laboration_4
Personen tillagd: Anders Nilsson
Kön: Male
Hår: Brunt, Kort
Födelsedatum: 1999-03-22 00:00:00
Tryck på enter för att gå vidare
Vill du lägga till en till person?
1: ja
2: nej


Run
Laboration_4
Dessa personer finns med på listan:
Anders Nilsson | Kön: Male | Född: 1999-03-22 00:00:00 | Hår: Kort, Brunt


Run
Laboration_4
****
Välj alternativ:
1: Lägg till ny person
2: Se tillagda personer
3: Avsluta programmet
****
3
Tack för att du har använt detta program
Process finished with exit code 0.

4. Diskussion och reflektion

4.1. Diskussion kring resultat

Kodningen gick generellt sett bra, jag övervägde olika sätt att kontrolera datumintputet innan jag bestämde mig för att fråga AI efter ett sätt att validera. En stor del av koden behövdes göras om vid valideringen för att kontrollen skulle göras korrekt, tidigare fanns det inga större checkar för att rätt sorts input accepterades, i slutändan valde jag att nestla switchsattser i ett try-catch block innuti en while loop för att fånga fel och endast tillåta korrekt inmatad data. Detta var nästan garanterat inte det absolut mest effektiva sättet att lösa problemet men det funkade bra. Överlag är jag nöjd med resultatet, programmet, funktionerna och felhanteringarna/valideringarna fungerar som de ska

4.2. Reflektion kring sprint 1

Sprint 1 gick utan större problem, på grund av de som tidigare nämnt specifika instruktionerna gick det snabbt och enkelt att implementera de olika kraven, jag skapade en enum för kön, struct för hår och en klass innehållande all relevant personlig information.

4.3. Reflektion kring sprint 2

I den andra sprinten var det stora problemet hur jag skulle gå tillväga för att skapa lösningar, vid flertalet moment valde jag att ändra, eller skriva om helt, koden för en funktion för att bättre lösa problemet och uppnå kraven. Det största exemplet på detta var hur programmet tog emot input från användaren då den skulle lägga till en person. Först var min tanke att ta input i form av en string och därefter validera om det var ett rimligt input givet vad som frågades efter. Jag insåg att detta inte var en bra lösning då det skulle krävas väldigt specifika checkar för att se till att strängen var passande, användaren skulle kunna mata in vad som helst. Jag valde därför att skriva om koden helt och istället ge användaren alternativ och ta emot input via en switchsats. Överlag gick sprint 2 bra.

4.4. Reflektion kring alternativa lösningar

Jag känner att det borde gå att göra valideringen ännu simplare, jag tror inte att min lösning nödvändigtvis var den mest effektiva, även om den funkade. Utöver detta skulle jag ha velat testa att bryta ner koden i mindre delar och göra om vissa delar till metoder för att undvika redundant kod

Frågor till AI-verktyg

Verktyg:

Fråga/prompt:

På vilket sätt svaret användes:

Verktyg: ChatGPT

Fråga/promt: “Jag gör ett program i C# och ska kontrollera ett input. Kan du hjälpa mig skapa ett kodblock som tar emot data från användaren angående ett datum och kollar om denna följer formatet “yyyymmdd””

På vilket sätt svaret användes: Jag var inte säker på hur man skulle validera formatet på ett datum, av ChatGPT fick jag denna kod:

```
validDate = DateTime.TryParseExact(
    input,
    "yyyyMMdd",
    CultureInfo.InvariantCulture,
    DateTimeStyles.None,
    out parsedDate
);
```

Vilken tar emot ett input från användaren och kollar om den matchar det önskade formatet genom att använda DateTime.TryParseExact, en metod som gör just detta, tar emot en sträng och kollar om den matchar ett givet format.

Referenser