

# Labbrapport: Labb 4

Författare: Jenny Wesström

Datum: 2026-01-09

**Commented [EE1]:** Uppdatera datum

Kursnamn: GIK299 – Objektorienterad Programmering

Examinator: Elin Ekman och Ulrika Artursson Wissa

## Innehåll

1.	Introduktion .....	3
2.	Metod.....	3
2.1.	Verktyg.....	3
2.2.	Stegvis beskrivning av tillvägagångssätt.....	3
2.3.	Förutsättningar för att göra labben .....	5
2.4.	Testning av koden.....	5
2.5.	Etiska överväganden.....	8
3.	Resultat .....	8
4.	Diskussion och reflektion .....	8
4.1.	Diskussion kring resultat .....	9
4.2.	Reflektion kring sprint 1 .....	9
4.3.	Reflektion kring sprint 2 .....	9
4.4.	Reflektion kring alternativa lösningar .....	9
	Frågor till AI-verktyg.....	10
	Referenser .....	12

## **1. Introduktion**

I den här labben har jag genomfört programmeringsuppgiften för labb 4 som gick ut på att skapa ett program för att samla, lagra samt addera information om personer i en lista med olika metoder. Syftet med denna rapport är att redovisa samt reflektera över genomförandet av labb 4. Jag kommer gå igenom metod, hur labben har genomförts, testning och felhantering, resultat samt reflektera och diskutera kring detta.

## **2. Metod**

### **2.1. Verktyg**

Verktyg som användes för att lösa labben

- Visual studio 2022, version 17.0 tillsammans med ramverket .Net 6.0 användes för att bygga och testa applikationen.
- Boken Pro C# with .NET 6
- Rikard Lands kompendium
- Tidigare labbar/lektionsexempel från canvas
- NUnit, version 3.13 användes för att testa koden

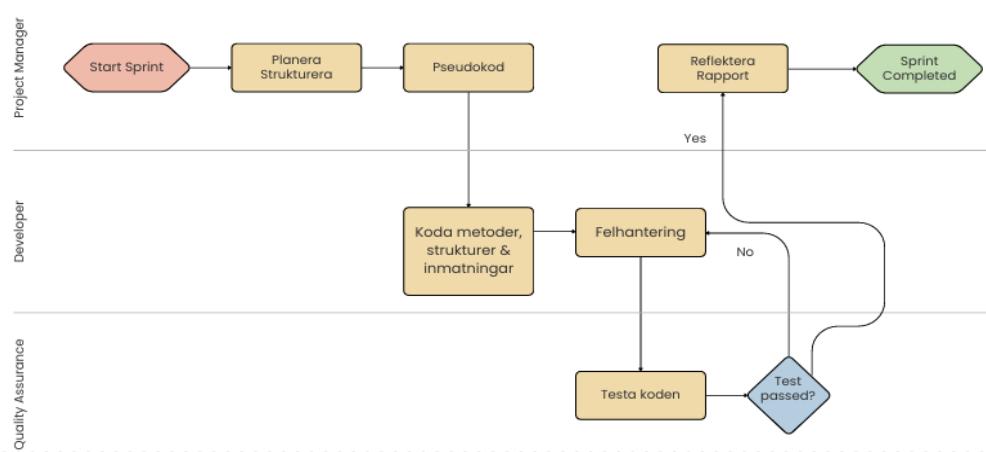
### **2.2. Stegvis beskrivning av tillvägagångssätt**

För sprint 1 började jag tillsammans med min labbpartner att enkelt och grovt bryta ned kravspecifikationen tillsammans för att få en förståelse av vad som skulle göras och i vilken ordning. Sedan gjorde vi en enkel pseudokod i Visual Studio exempelvis ”// create properties for eye colour and date of birth ” samt ”// create a ToString method ”.

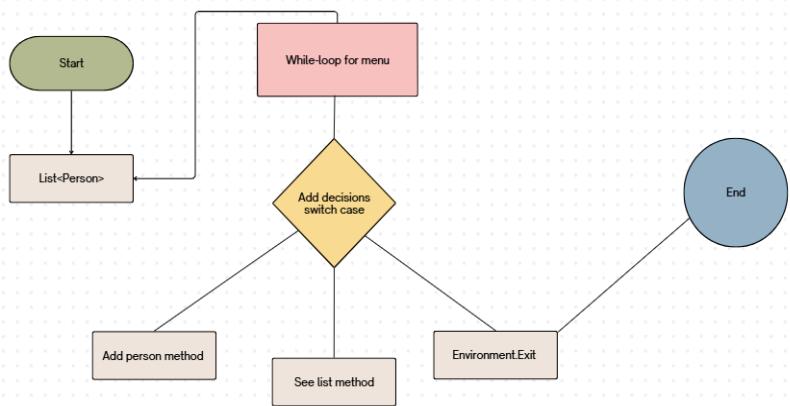
Vi utgick ifrån kravspecifikationen och dess ordning, först skapade vi en enum som vi döpte till ”Gender.cs” sedan skapade vi en struct, ”Hair.cs” och sedan en klass som vi döpte till ”Person.cs”. Därefter skapade vi properties för Person-klassen, konstruktor samt ToString-metoden. Vi genomförde hela sprint 1 tillsammans under en session, men med varsin dator för att båda skulle få arbeta och lära sig då vi bedömde att den första sprinten var relativt liten i omfattning.

Inför sprint 2 blev jag av med en labbpartner och tog med mig en lärdom från sprint 1, att planera/strukturera samt bryta ned arbetet i mindre delar. Jag började med att göra ett mycket grovt flödesschema för hela sprinten för att strukturera upp arbetet och få en bättre överblick över vad som skulle göras. Sedan gjorde jag en enkel pseudokod i Visual Studio där jag skrev ut kommentarer utifrån kravspecifikationen. Därefter arbetade jag systematiskt utifrån kravspecifikationen.

## Labb4 Sprint 2



## Sprint 2



Flödesscheman skapade av mig i verktyget Canva (Canva, 2026)

### 2.3. Förutsättningar för att göra labben

För att kunna genomföra labben så användes Visual Studio och .NET ramverket samt Canva, ett online-verktyg för att skapa flödesscheman (Canva, 2026). Jag använde mig även av tidigare labbar samt lektionsexempel i canvas.

### 2.4. Testning av koden

Testning av koden genomfördes delvis löpande genom att köra koden regelbundet efter att nya metoder och block kodats. Sedan planerades även att mer djupgående testa koden genom att genomföra enhetstester med felaktiga inmatningar på alla delar av programmet. Jag gjorde detta genom att ta en del av programmet och skriva ned möjliga felaktiga inmatningar jag kunde komma på. I denna process kom jag på fler felaktiga inmatningar ju fler delar jag testade och var tvungen att gå tillbaka och börja om för att kontrollera tidigare testade enheter. Detta gjordes först efter att hela den grundläggande strukturen och gränssnitt var färdigt. Därefter byggdes programmet ut och i vissa delar byggdes det om helt för renare kod och mer robust felhantering.

Det som byggdes om helt var att jag valde att skapa en egen klass för validering (InputValidator) av inmatningar med metoder för felhantering av alla de olika inmatningarna i programmet. Jag lade även till en while-loop då jag insåg att koden på flera ställen vid felinmatning gick tillbaka till den yttre while-loopen och jag ville se till att användaren skulle kunna fortsätta med inmatningar för ett visst menyval tills inmatningen blev korrekt.

Här testades metoden "GetValidHairColour()" med felaktiga inputs då jag testade 1. Siffror 2. Tom input 3. En bokstav 4. Ett ord 5. Rätt input men med stora bokstäver. Test 1–4 fungerade, men stora bokstäver

```
C:\Users\wesss\OneDrive\C#F + 
*Hair length*
---short, medium or long---
Enter hair length: short

!!! -Invalid Input- !!!

*Hair length*
Must be:
short, medium or long.

### Try again! ###

*Hair length*
---short, medium or long---
Enter hair length: SHORT

!!! -Invalid Input- !!!

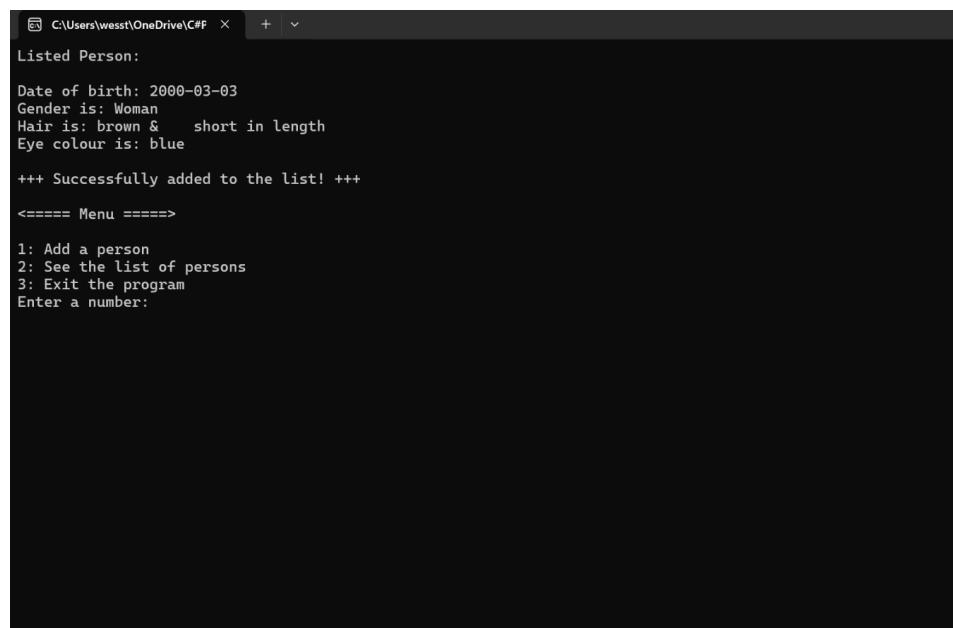
*Hair length*
Must be:
short, medium or long.

### Try again! ###

*Hair length*
---short, medium or long---
Enter hair length: |
```

gav felmeddelande och det hade jag förväntat mig då jag kom på detta när jag planerade enhetstestet. Detta lösades med metoden ".ToLower()" på input-satsen för att konvertera inputen till små bokstäver. Detta lades även till på de andra valideringsmetoderna med string som datatyp även om det inte krävdes för rätt input, men för en snyggare samt mer enhetlig utskrift av listade personer.

När jag la till denna metod så kom jag även på metoden ".Trim()" från Microsofts introduktionskurs till c# på Youtube (dotnet, 2023). Denna metod var inte nödvändig för valideringen att fungera, men för att garantera en snygg utskrift lades den till. Metoden tar bort "whitespace" (mellanslag) både före och efter strängen.



```
C:\Users\wesst\OneDrive\C#> + ^

Listed Person:

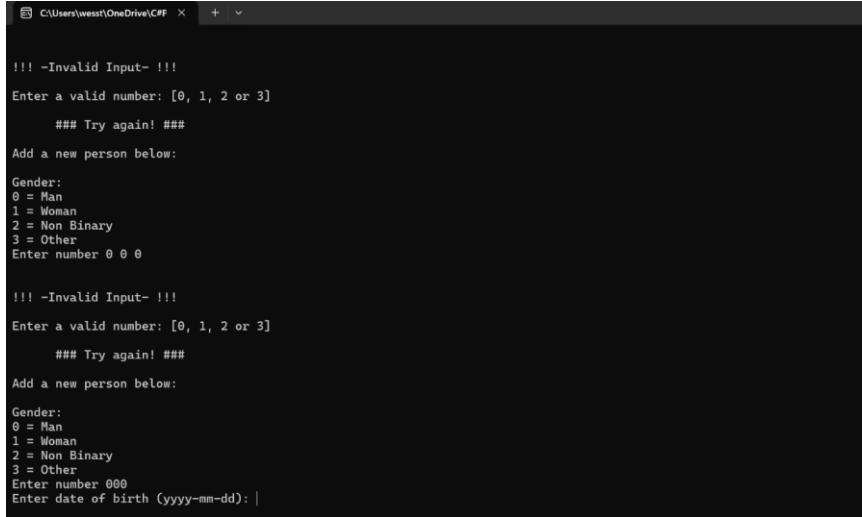
Date of birth: 2000-03-03
Gender is: Woman
Hair is: brown &      short in length
Eye colour is: blue

+++ Successfully added to the list! +++

<===== Menu =====>

1: Add a person
2: See the list of persons
3: Exit the program
Enter a number:
```

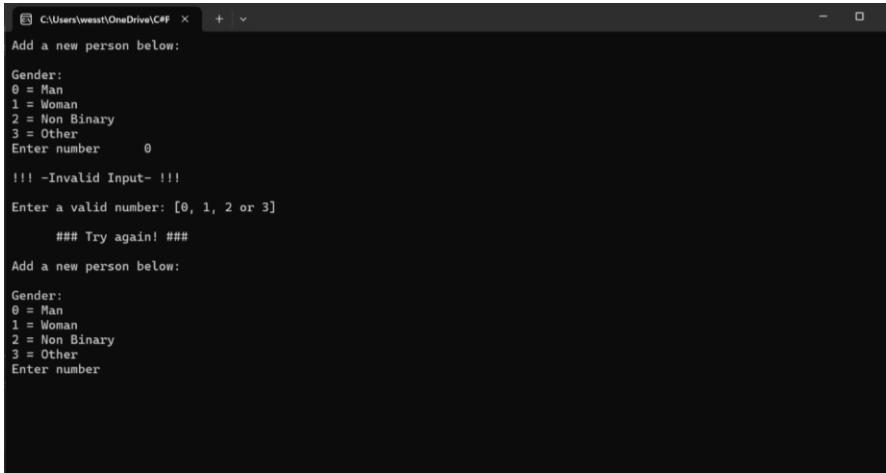
Skärmbild på körning där inputen "short" har gjorts med flera mellanslag före.



```
!!! -Invalid Input- !!!  
Enter a valid number: [0, 1, 2 or 3]  
### Try again! ###  
Add a new person below:  
Gender:  
0 = Man  
1 = Woman  
2 = Non Binary  
3 = Other  
Enter number 0 0 0  
  
!!! -Invalid Input- !!!  
Enter a valid number: [0, 1, 2 or 3]  
### Try again! ###  
Add a new person below:  
Gender:  
0 = Man  
1 = Woman  
2 = Non Binary  
3 = Other  
Enter number 000  
Enter date of birth (yyyy-mm-dd): |
```

Skärmbild från körning som visar att input "000" går igenom som valid.

En intressant observation som uppkom under "GetValidGenderInput()" var att 000 gick igenom som valid. "111" eller "222" gick inte igenom även "0 0 0" gick inte igenom, jag förstod att det hade med intervallet att göra då 000 är inom 0-3 intervallet, men 0 0 0 innehåller mellanslag. Det tog mig lång tid att lösa denna, men efter att ha brutit ned det till vad jag vill kolla till: "längd på inputen == 1" så var lösningen enkel. La till "input.Length == 1" i if-satsen. Det löste ett problem men eventuella mellanslag i inputen före eller efter strängen resulterade i felaktig input då strängen blev längre än 1. Detta löstes med ".Trim()"-metoden.



```
Add a new person below:  
Gender:  
0 = Man  
1 = Woman  
2 = Non Binary  
3 = Other  
Enter number 0  
  
!!! -Invalid Input- !!!  
Enter a valid number: [0, 1, 2 or 3]  
### Try again! ###  
Add a new person below:  
Gender:  
0 = Man  
1 = Woman  
2 = Non Binary  
3 = Other  
Enter number
```

Skärmbild från körning som visar att input "0" är invalid.

Jag lärde mig även lite grundläggande om NUnit-tester och genomförde tillsammans med full guidning AI-verktyget Copilot tester för programmet.

## 2.5. Etiska överväganden

Eftersom detta program inte hanterade verkliga personuppgifter så var etiska övervägande, GDPR eller andra lagar inte nödvändigt att ta hänsyn till.

## 3. Resultat

Till slut så lyckades jag implementera funktionaliteten helt enligt kraven och med en renare kod än i tidigt stadio. Det första utkastet fungerade, men hade definitivt brister i felhanteringen samt såg rörigt ut i program.cs filen med väldigt mycket kod. Koden var hårdkodad med många nästlade if-satser som gjorde koden svår att läsa och underhålla.

Genom att skapa en ny valideringsklass (InputValidator) fick jag också en möjlighet att se om det gick att koda felhanteringen på ett annat eller till och med bättre sätt. Det var egentligen inget problem, programmet fungerade som det såg ut, men det ledde till ett bättre slutresultat, mer robust, läsbar, hanterbar samt skalbar kod.

Sprint 2 var enligt mig mer omfattande samt blev något rörigt då jag först hårdkodade ett funktionellt program enligt kravspecifikationen för att sedan inse att det går att designa detta bättre vad det gäller läsbarhet, skalbarhet och underhållbarhet. Testningen av koden som genomfördes löpande bidrog till att identifiera och lösa problem under utvecklingen exempelvis en slarvig felkodad intervall för if-sats.

## 4. Diskussion och reflektion

Jag är nöjd med resultatet och processen även om den hade kunnat förbättrats. Den största insikten/lärdomen från denna uppgift är nog främst att det är en iterativ process, det är helt okej att först koda ett fungerande program med brister för att sedan utveckla det stevvis. Det kanske rent av är en naturlig del av processen att lära sig att se vart det går att utveckla och förbättra koden.

En annan lärdom som jag tar med mig är vikten av det förberedande arbetet med hjälp av flödesscheman, UML-diagram samt product backlog. Framför allt product backlog och UML-diagram som jag inte använde för den här labben, men lärde mig om parallellt med ett annat projekt. Hade jag gjort om denna labb så hade jag använt mig av product backlog och UML-diagram tillsammans med kanban för mer struktur.

Vad det gäller den tekniska delen så har jag främst lärt mig mer om principer för ökad kodkvalitet, att skriva renare kod, mer skalbar samt underhållbar kod. Jag har även lärt mig litegrann om hur man skapar NUnit tester och varför det är bättre än att testa koden manuellt genom att köra den i konsolen.

#### **4.1. Diskussion kring resultat**

Slutresultatet anser jag blev som förväntat utifrån kunskapsnivån idag, är helt säker på att det går att förbättra exempelvis felhanteringen, men även generellt läsbarheten med renare kod, samt skalbarheten och underhållsbarheten.

#### **4.2. Reflektion kring sprint 1**

Sprint 1 genomförde jag tillsammans med en tidigare student, det gick bra, vi gjorde dock inget flödesschema då men vi bröt ned uppgifter utifrån kravspecifikationen samt gjorde pseudokod i Visual studio. Det jag tar med mig från sprint 1 är att jag gärna gör flödesschema och tydlig planering för att enklare ha en översikt över arbetet samt struktur.

#### **4.3. Reflektion kring sprint 2**

För sprint 2 så gjorde jag ett övergripande flödesschema samt en planering kring kravspecifikationen. Detta gjorde jag även löpande under kodningen genom att skriva ned vad som behöver göras närmast 3–4 punkter och arbeta utefter det. Under det arbetet framkommer givetvis ofta nya ”problem” eller tankar kring lösningar som kan implementeras så to do-listan uppdaterades löpande.

Sprint 2 var betydligt mer omfattande då sprint 1 var mer grunden, strukturen och så här i efterhand så förstår jag att en tydligare planering med product backlog samt kanban hade varit givande.

#### **4.4. Reflektion kring alternativa lösningar**

Under sprint 2 så insåg jag snabbt att jag hårdkodade väldigt mycket och byggde om en del av koden där jag såg att det gick att göra bättre. Har under tiden för arbetet även kikat på andra studenter som har betydligt renare kod så hade jag fått göra om labben så hade jag absolut lagt tid på att lära mig andra sätt att designa programmet på. Jag försökte mig även på try/catch för felhantering, men kände att jag dels inte kan det tillräckligt väl, dels bedömde jag att det inte behövdes för denna labb. Min uppfattning efter att ha läst på om try/catch på Microsofts .NET dokumentation (Microsoft, 2023) så är det mer relevant i större program där det bedöms finnas oförutsedda fel, men det är absolut något jag hade velat lära mig mer om och använda om jag skulle göra om labben.

## Frågor till AI-verktyg

Verktyg: Copilot

**Fråga/prompt:** Jag klistrade in en squiggle där jag hade skrivit if (eyeColour != string ...)

**På vilket sätt svaret användes:** Fick som svar att jag kan explicit kan skriva "is not", men jag visste att jag kan använda utropstecknet och svaret hjälpte mig att komma ur hjärnsläppet då "is not" är vad "!" står för så då insåg jag att det som blivit tokigt var lika med tecknet.

Verktyg: Copilot

**Fråga/prompt:** string input = Console.ReadLine(); // CS8600 warning here what is it and how do I solve it? What is best practice?

**På vilket sätt svaret användes:**

Svar:

"Why? Because `Console.ReadLine()` can return null (e.g., if the input stream is closed), but `string` input is non-nullable by default in modern C#.

How to fix it

You have two main options:

Option 1: Allow nulls (nullable type)

`string? input = Console.ReadLine();`

This tells the compiler: "Yes, input might be null, and I'm okay with that."

Option 2: Add a null check

`string input = Console.ReadLine() ?? "";`

This says: "If `Console.ReadLine()` returns null, use an empty string instead."

 Best practice for user input

For `InputValidator`, I recommend Option 2 most of the time, because you usually want to avoid nulls and treat empty input as invalid:..."

**Fråga/prompt:** "Okay but what difference will it make? Because I already have a check for null or white space with `.string.IsNullOrWhiteSpace`. what will happen if I dont use a null check on the input line?"

Svar:

 **Without the null check**

Suppose you wrote:

```
string input = Console.ReadLine(); // no ?? ""  
if (string.IsNullOrWhiteSpace(input))  
{  
    // handle invalid input  
}  
    • Console.ReadLine() can return null.
```

- `string.IsNullOrEmpty(null)` is actually safe — it returns true if the argument is null.  
So your program wouldn't crash here.

---

### **⚠️ But the compiler doesn't know that**

- Because your variable is declared as `string` (non-nullable), the compiler warns:  
*"You're assigning something that might be null to a non-nullable type."*
- That's the CS8600 warning you saw.
- It's not a runtime error — it's the compiler protecting you against possible null reference bugs elsewhere in your code.

---

### **✅ With the null check (`?? ""`)**

By writing:

```
string input = Console.ReadLine() ?? "";
```

you guarantee that `input` is **never null**.

- The compiler is happy (no warnings).
- Your later code (like `IsNullOrEmpty`) doesn't have to worry about nulls.
- You avoid subtle bugs if you ever pass `input` into a method that *doesn't* handle null safely.

---

### **❖ Key takeaway**

- **Without null check:** Safe in this specific case, but compiler warns because `string` isn't supposed to hold null.
- **With null check:** Cleaner, safer, and future-proof — no warnings, no risk of accidentally passing null somewhere else. “

**På vilket sätt svaret användes:** Jag valde då att använda "null check" framför allt för att ha en säkrare, mer robust kod att kunna bygga vidare på utan risk för att skicka vidare "null".

## Referenser

Canva (2026). *Canva*. <https://canva.com/>

Dotnet (10 november 2023). *C# for Beginners*. [video] YouTube.  
<https://youtu.be/9THmGiSPjBQ?si=y0IRwzTtuW8ldf2N>

Land, R. (19 november 2024). *Introduktion till programmering*. [Kompendium] (1 uppl.) Högskolan Dalarna. <https://canvas.du.se/>

Microsoft. (2023). *C# Guide and .NET Documentation on Microsoft Learn*. Microsoft Learn.  
<https://learn.microsoft.com/en-us/dotnet/csharp/>

Troelsen, A., & Japikse, P. (2022). *Pro C# 10 with .NET 6*. (11 uppl.). Apress.