GIK299: Objektorienterad programmering
Data och Informationshantering
Institutionen för information och teknik
Högskolan Dalarna

Laborationsrapport
HT 2024

Labbrapport: Labb 4

Författare: Christofer Johansson och Emil Blom

Datum: 8 januari 2026

Kursnamn: GIK299 - Objektorienterad Programmering

Examinator: Elin Ekman och Ulrika Artursson Wissa

Innehåll

1 Introduktion

I följande laboration har vi skapat ett program i c# med målet att samla information om en person för att sedan spara detta på ett effektivt och smart sätt. Vi har fått strikta instruktioner för hur datan ska samlas och hanteras.

2 Metod

2.1 Verktyg

- Visual Studio 2026
- Github
- .net10.0
- Exempelkod från Elin
- laTex genom Overleaf
- Vi har använt oss givna instruktioner ifrån Labb instruktioner
- Vi har utgått ifrån givna Labb mallen
- Vi har utgått ifrån givna Parprogrammerings mallen
- Vi har vid rådgivning använt GenAI Gemini
- Vi har även vid felsökning använt GeeksforGeeks

2.2 Stegvis beskrivning av tillvägagångssätt

Under hela arbetetstiden har vi suttit på discord och skärmdelat våra IDEs till varandra, detta har lett till att vi lätt kunnat hjälpas åt.
Vi delade upp arbetet och Emil skapade Person.cs, Program.cs och doB.cs. Christofer skapade Gender.cs, Hair.cs och EyeColor.cs.
I Person.cs har vi utgått mycket ifrån Book.cs som vi fått tillgång till. Där samma struktur ses där vi först listar upp Properties för att sedan i en konstruktor initierar våra variabler för att sedan spara de i en lista.
Därefter skapades Program.cs som även det bygger mycket på Program.cs. Vi börjar med att initiera de variabler vi kommer använda, i detta fallet är det:

```
int day, month, year;
string chosenColor = HairCreation.CreateHairColor();
int chosenLength = HairCreation.CreateHairLength();

GenderCreation.CreateGender();
EyeCreation.CreateEye();

Hair haircolor = new Hair(chosenColor, chosenLength);
Gender gender = new Gender { };
Eyecolor eyecolor = EyeCreation.CreateEye();
DoB.SelectDoB(out day, out month, out year);
```

Programkod 1: *Variabler som initieras och inputs spars ifrån våra metoder*

Inuti while loopen använder vi switch cases för de möjliga valen. Notera att våran default skickar tillbaka oss ifall vi skriver fel inputs.

```
Person newPerson = new Person(haircolor, gender, eyecolor, day,
    month, year);
Person.AddPerson(newPerson);
```

Programkod 2: *Här kombinerar vi informationen som samlats från metoder för att skapa en ny person och lägga till i en lista*

doB.cs är för att ta in födelsedagsdatum, detta görs genom en funktion där vi ber om 3 interger intputs för dag, månad och år.
Gender.cs fanns det krav på att det skulle vara ett enum och vi har givetvist följt detta genom följande kodsnutt (där vi även satt Man till 1 på index för att inte starta på 0):

```
public enum Gender
{
    Man = 1,
    Woman,
    NonBinary,
    Other
}
```

Programkod 3: *Enum för kön*

Sedan har vi gjort en funktion för att fråga om input från användaren om deras kön, vi väljer att använda foreach för att gå igeom alla kön och rada upp valen användaren har. I slutändan använde vi oss på ett liknande sätt utav foreach-loopar vid input av nästan alla attribut. Se kodsnutt för gender nedan:

```
public class GenderCreation
{
    public static void CreateGender()
    {
        Console.WriteLine("Please␣select␣one␣of␣the␣following␣
            genders");
        int current = 1;
        foreach (Gender g in Enum.GetValues(typeof(Gender))) //
            Loops through all enum values and we can add more
            genders without changing this code
        {
            Console.WriteLine($"{current}.␣" + g);
            current++;
        }
        Console.Write("Select␣(1-4):␣");


        while (true)
        {
            string input = Console.ReadLine();

            // Check if a valid number was entered
            if (Enum.TryParse(input, out Gender chosenGender) &&
                Enum.IsDefined(typeof(Gender), chosenGender))
            {
                Console.WriteLine($"You␣selected:␣{chosenGender}")
                    ;
                break;
            }
            else
            {
                Console.WriteLine("Invalid␣selection.");
            }
        }
    }
}
```

Programkod 4: *Gender funktionen, notera att ␣ är mellanslag*

För filen Hair.cs så har vi fått krav på att det ska vara ett struct och här använder vi properties återigen för våran hårfärg och hårlängd. För att sedan kunna få ut detta värden och använda dem har vi använt public override string ToString(), se följande kodsnutt:

```
public struct Hair
{
    public string HairColor { get; set; }
    public int HairLength { get; set; }
    public Hair(string color, int length)
    {
        HairColor = color;
        HairLength = length;
    }
    public override string ToString()
    {
        return $"Hair␣Color:␣{HairColor}␣" + $"\nHair␣Length:␣{
            HairLength}cm";
    }
}
```

Programkod 5: *Hair som ett struct, notera att ␣ är mellanslag*

Sedan för att kunna göra val och stoppa in värden i struct Hair så har vi 2 funktioner, en för hårfärg och en för hårlängd, Se följande kodsnutt:

```
public static string CreateHairColor()
{

    string[] HairChoice = { "Black", "Brown", "Blonde", "Red", "
       White", "Multicolor" };

    Console.WriteLine("Please␣select␣one␣of␣the␣following␣Hair␣
       colors");
    int current = 1;
    foreach (string g in HairChoice)
    {
        Console.WriteLine($"{current}.␣" + g);
        current++;
    }
    Console.Write("Select␣(1-6):␣");


    while (true)
    {
        string input = Console.ReadLine();
        int.TryParse(input, out int choice);

        // Check if input is of the correct type while also
           checking if it is within the allowed range
        if (choice == 1 || choice == 2 || choice == 3 || choice ==
            4 || choice == 5 || choice == 6)
        {
            Console.WriteLine($"You␣selected:␣{HairChoice[choice
               -1]}");
            return HairChoice[choice];
        }
        else
        {
            Console.WriteLine("Invalid␣selection.");
        }
    }
}
```

Programkod 6: *första funktionen där vi hämtar färg, notera att ␣ är mellanslag*

Andra funktionen, här ber programmet om ett input från användaren för längd, vi väljer även här att sätta en max gräns på 300cm och en minsta gräns på > 0.

```
public static int CreateHairLength()
{

    Console.WriteLine("Please␣write␣out␣ur␣hair␣length␣in␣cm:␣");

    while (true)
    {
        string input = Console.ReadLine();
        // Check if input is of the correct type while also
           checking if it is within the allowed range
        if (int.TryParse(input, out int choice) && choice > 0 &&
           choice < 300) // Sets a logical max range
        {
            Console.WriteLine($"You␣selected:␣{choice}cm");
            return choice;
        }
        else
        {
            Console.WriteLine("Invalid␣selection.");
        }
    }
}
```

Programkod 7: *andra funktionen där vi hämtar längd, notera att ␣ är mellanslag*

Ögonfärg var något som vi först gjorde mycket svårare än det behövde vara, vi lade även till heterokromi vilket är när en person har 2 olika ögonfärger, men beslutade oss för att skippa detta då det visade sig vara svårare än vi först trott att implementera.

```
public class EyeCreation
{
    public static Eyecolor CreateEye()
    {
        Console.WriteLine("Please␣select␣one␣of␣the␣following␣
            eyecolors:");
        int current = 1;
        foreach (Eyecolor c in Enum.GetValues(typeof(Eyecolor)))
        {
            Console.WriteLine($"{current}.␣" + c);
            current++;
        }
        while (true)
        {
            Console.Write("Select␣(1-5):␣");
            string input = Console.ReadLine();
            // Check if a valid number was entered
            if (int.TryParse(input, out int choice) && choice >= 1
                && choice <= 6)
            {
                Eyecolor selected = (Eyecolor)choice;
                Console.Clear();
                return selected;
            }
            else
            {
                Console.WriteLine("Invalid␣selection.");
                Console.WriteLine("----------");
            }
        }
    }
}
```

Programkod 8: *funktionen vi använt för att be om ögonfärg, Eyecolor är ett enum som nämns tidigare i koden, notera att ␣ är mellanslag*

2.3 Förutsättningar för att göra labben

Vi har programmerat i Visual Studio med frameworket .net10.0.

2.4 Testning av koden

Vi har planerat koden på ett sådant sätt där de vanligaste förekommande felen (skriva in bokstav när vi ber om siffra etc) inte är ett problem då TryParse har använts överallt i en while loop så vid fel input kommer loopen att börja om för att be om ett nytt input.

Utöver det har vi även valt att inte använda explicita bools för våra while-loopar, utan använder i stället while (true) eftersom loopen enkelt avslutas när ett värde returneras, i stället för att kräva att en bool sätts till false eller blir bruten (break). Vi har även vid hårlängd satt en max och min gräns på 300cm respektive 0cm för att det ska vara verklighetstroget.

2.5 Etiska överväganden

Vi har enbart använt fiktiva personer när vi har testat koden, du som användare har ett ansvar att använda den på ett ansvarsfullt vis. Skulle det vara så att detta eller ett liknande program skulle användas komnersiellt skulle stor vikt behövas läggas på att datan som samlas lagras på ett säkert sätt med hjälp av exempelvis kryptering.

3 Resultat

Vi har utgått från instruktionerna givna av evil corp och lyckades genomföra upgiften med stor framgång. Vi har lyckats väl med själva logiken, men det fanns många hinder längs vägen Ett problem vi upptäckte tidigt var att vid klassanvändning kunde koden lätt bli rörig om vi inte var väldigt noga med själva planeringen och strukturen av koden.

4 Diskussion och reflektion

Den röriga strukturen vi upplevde har nog mycket att göra med att vi inte gjorde något flödesschema, vilket vi i efterhand ångrar rejält. Trots ett saknat flödesschema lyckades vi ändå till slut att få ihop en läsbar kod som fungerar bra. Det krävdes dock lite omstruktur för att åstadkomma detta, vilket man kan ses som tidsmässigt ineffektivt. Det vi skulle kunna göra för att gå vidare är spara infon vi samlar i en lista i exempelvis .txt format så att informationen spars efter programet avslutas. Det känns som att varje projekt kan bli svårt att hantera tidsmässigt för att det går hela tiden att lägga till mer och mer funktionalitet och livskvalitet. Det absolut viktigaste vi tar ifrån oss är hur viktigt planering är, att koda och felsöka känns som den lättare delen medans en bra plan och en bra ide för vad som efterfrågas är minst lika viktigt som att skriva ut koden.

4.1 Diskussion kring resultat

Det var svårt till en början med olika filer och att hålla reda på variablerna, struct, enum och metoderna och vad alla heter, vilka och vad som är public och inte men vi har ändå lyckas uppnå ett resultat där vi båda är nöjda med koden.

4.2 Reflektion kring sprint 1

Ett bättre planering behövs då vi gick in med mentaliteten av att vi testar och ser hur det går och hur långt vi kommer. Inför sprint 2 ska vi ha ett bättre planerande, vad

kommer göras i vilken fil samt planera in ifall vi inte hinner klart allt på en dag i sprint 2.

4.3 Reflektion kring sprint 2

Återigen så är vi lite för ivriga och börjar koda innan vi riktigt har prata oss igenom vad vi faktiskt ska göra och hur vi ska göra det. Sprint 2 tog betydligt längre tid än tänkt vilket troligen är på grund av våran dåliga planering och svårigheten att jobba med objektorienterad programmering i åtanke. Trots dessa svårigheter lyckades vi ändå med arbetet och lärde oss mycket om objektorienterad struktur och har nu även bättre förståelse när det gäller listor, enums och liknande.

4.4 Reflektion kring alternativa lösningar

Vi kan förstå hur enums och structs kan vara bra, men om vi skulle göra om det hade vi bara gjort en metod per efterfrågad attribut. Kanske att man fått skriva in själv kön, hårfärg och sen jämfört med med listan i enum om det finns ett sådan kön och om det finns i enum så går man vidare i koden annars ber den om input igen.

5 Frågor till AI-verktyg

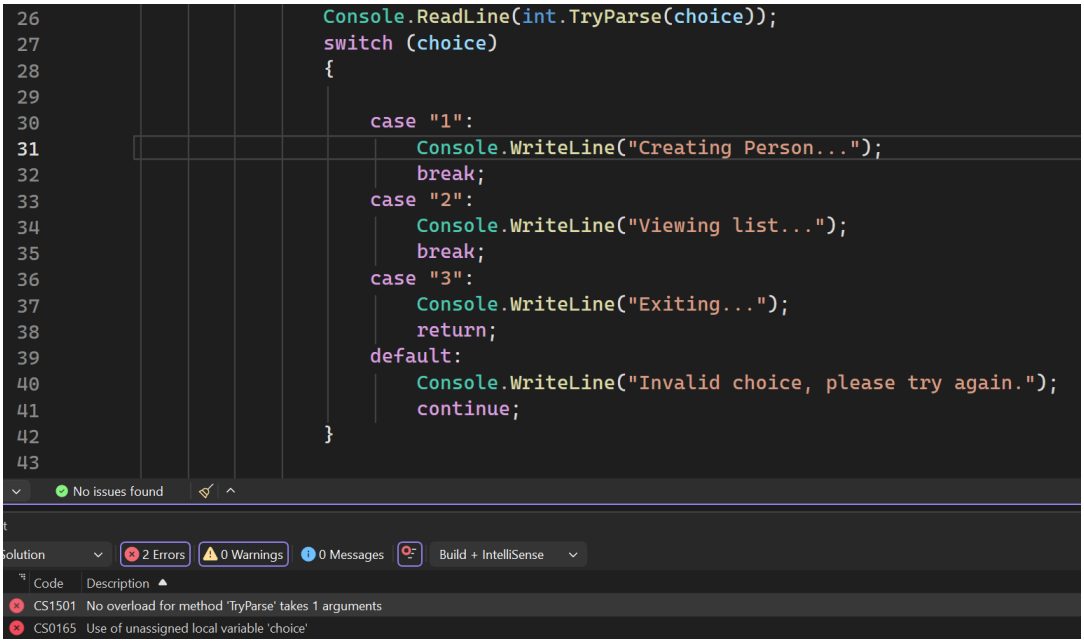
Verktyg: Gemini
Fråga/prompt: What is a good format for a birthday? And do I can I convert a 4-digit year input to 2-digit?
På vilket sätt svaret användes: Svaret användes för att komma fram till DD-MM-YY formatet samt hur man går från 4-digit till 2-digit (year

Verktyg: Gemini
Fråga/prompt: How can I make sure the current year stays updated (doesn't get stuck at 2025)
På vilket sätt svaret användes: Svaret användes för att, med hjälp av DateTime.Now kunna se till att date of birth-valet alltid är up to date, utan att man behöver ändra koden varje år och särskilt på skottår.

Verktyg: Gemini
Fråga/prompt: varför är inte if (int.TryParse(input, out int choice) && choice 0 < 300) rätt?
På vilket sätt svaret användes: Man behövde dela upp det i två olika jämföringar, så la till && choice > 0 && choice < 300 och det används i koden Hair.cs

Verktyg: Gemini
Fråga/prompt: Vad gör det här public override string ToString()
På vilket sätt svaret användes: Jag (Christofer) ville ha bättre förståelse för override string ToString() då vi använder det vid struct i Hair.cs.

Verktyg: Gemini
Fråga/prompt: wtf is overload for method try parse



Figur 1: *Bilden jag bifogade till gemeni tillsammans med frågan*

På vilket sätt svaret användes: Lärde mig om return 0; och return ; för att kringå att vi inte alltid har en output, men vi har alltid en output för vi har failsafes som kommer in innan den kan returna inget. Koden används i Hair.Cs

6 Bilagor

Parprogrammeringslogg

Tabell 1: *Logg över Parprogrammeringstimmar*

Datum	**Tid i timmar**	**Ensam eller i par**	**% fördelat i att sitta vid tangentbordet**
2025-12-05	4	I par	50%
2025-12-18	5	I par	50%
2025-12-18	2	Christofer	100%
2025-12-18	3	Emil	100%
2026-00-08	1	Christfer	100%
Totalt:	15	I par	50%