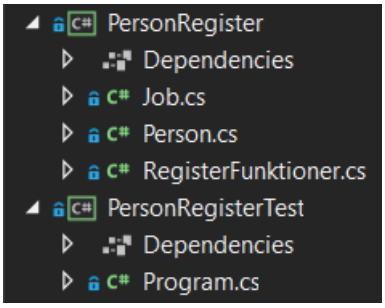
Libraries

Da jeg først hørte libraries tænkte jeg det var meget mere kompliceret end det egentlig er. Men så da jeg først selv fik fingrene ned i det fandt jeg hurtigt ud af. Et library kunne bruges som en slags skabelon eller ligesom det ligger i ordet et bibliotek, et sted hvor man henter information. Man kunne fx lave et library til at lave en person, hvor man i librariet allerede på forhånd har oprettet en masse variabler og funktioner man kan bruge til at beskrive en person med. Sådan man ikke skal oprette alle variablerne hver gang man skal bruge dem i nye projekter.

```
public class Person
{
    public string name;
    public int age;
    public Job job;
    public string nationality;
}
```

Personregisterprojektet

Personregisterprojektet var en god starter opgave synes jeg fordi, vi blev sat ind i at arbejde med objekter og classes med det samme. Jeg føler selv jeg lærte en masse omkring hvorfor man ligesom vil bruge classes i stedet for at skrive alt i et Main script. Det var meget mere overskueligt når man ligesom havde en oversigt over hvor alt var henne af. Og man så derefter kunne oprette et test script til at se om det faktisk virkede.



Dictionaries

Da jeg først så hvordan dictionaries virkede så tænkte jeg, hvorfor man ikke bare ville bruge lister eller arrays. Da jeg så fik arbejdet lidt med dem i mit personregister, kunne jeg lige pludselig godt se hvorfor man vil bruge dictionaries i nogle projekter over arrays og lister. Fx hvis man arbejdede med en masse brugere så i stedet for, skulle finde deres position i arrayet eller listen så kunne man bare skrive et keyword ind som fx et id eller et navn.

```
funktioner.navnOgAlderDic["Timmy"] = 33;
funktioner.navnOgAlderDic["Peter"] = 41;
```

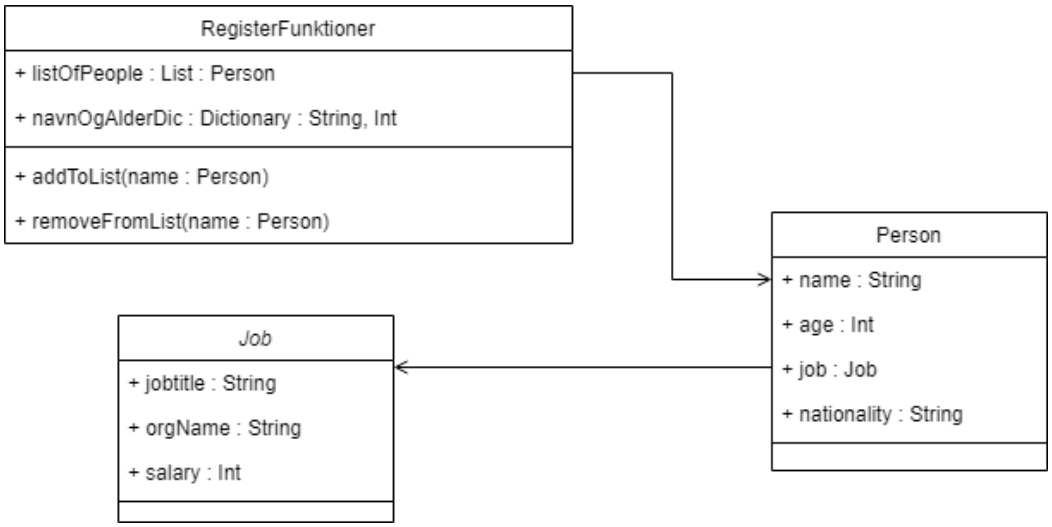
Det her er et eksempel hvor jeg har et dictionary med navnet ”navnOgAlderDic”, som er oprettet i objektet ”funktioner”. Der bliver først givet keywordet ”Timmy” hvorefter hans alder bliver givet til det keyword så når man skriver ”Timmy” ind i systemet er det hans alder, der bliver vist som er 33.

```
Timmy
Timmy is 33 years old.
```

```
Console.WriteLine($"{"Timmy"} is {funktioner.navnOgAlderDic["Timmy"]} years old.")
```

UML Class Diagrams

UML class diagrammer er nok noget af det bedste jeg har brugt til at planlægge et projekt eller få overblik over et projekt, som allerede er i gang. Jeg synes det var et rigtigt godt værktøj til det personregisterprojekt som allerede var oprettet så man ligesom kunne holde ordentligt styr på hvilke ting man overhovedet fik brugt til noget og de ting der aldrig blev brugt kunne fjernes.



Det gav også et godt indblik i hvordan ting kan ændre sig undervejs når man indser ting man ikke fik brug for. Det oplevet jeg i hvert fald i skolesystem projektet hvor vi skulle lave et diagram også lave koden efterfølgende der var ligesom nogle af de parametre, man havde skrevet ind i diagrammet som ikke var nødvendige under kodningen fordi, man lige kom i tanke om en nemmere eller smartere løsning. Det er også en af fordelene ved at have lavet det inden man koder, det er så nemt bare lige at gå ind og ændrer i diagrammet hvis du ændrer mening undervejs.

Overloading

Jeg synes overloading er en fiks lille ting der bestemt kan være rar at kende til når man arbejder med constructers. Så man kan lave undtagelser i ens program hvis du nu skal assigne et arbejde og du ikke har et så den bare bliver sat til en default værdi såsom ”Unknown”.

```
public Person(string name, int age)
        {
            this.name = name;
            this.age = age;
            job = "Unknown";
        }
```

MathF:

Så var der også et lille underemne jeg kiggede lidt på da vi skulle lave overloading med plus, minus, gange, dividere osv. Som var Mathf som jeg brugte til at lave kvadratrod og potens regning med det

var super nemt og lige til hvis man skulle finde kvadratroden af noget skulle man noget så nemt som at skrive:

```
public float kvadratrod(float a)
        {
            return MathF.Sqrt(a);
        }
```

Når man skriver Sqrt(a) så er det ligesom at sige ”find the square root of (a)”.

Delegates/Lambda Expressions

Da vi først arbejdede med delegates var jeg på bar bund jeg forstod ikke rigtig hvordan det fungerede. Så da vi skulle til at indsætte dem i vores matematik overloading projekt så gav det hele lige pludselig mening at man kunne lave en delegate som gemmer den i en variabel som du kan kalde efterfølgende med det angivet navn.

```
Delegates.firstDel del1 = Delegates.target;
del1();
```

Det jeg personligt selv synes der virkelig var smart er lambda expressions som kunne bruges, hvis der ikke var nogen target funktion så ville man oprette funktionen inde i lambda expressionen.

```
Delegates.LambdaExp expression = (int a) => a*2;
Console.WriteLine(expression(4));
```

I det her eksempel der bliver der givet en værdi som er ”a” så tager den og ganger a med 2. Hvorefter i konsollen der bliver kaldt ”expression” med værdien 4 så i konsollen der bliver printet 8.