**Beskrivelse af opgaven:**

Lav et C# applikation til at oprette en eller flere liste-struktur der kan indeholde objekter af en eller flere klasse datastruktur.

Data på en given person kan angives ved navn, alder, bopal, postnummer, job-og-løn, telefon nummer eller skole/universitet mm. Personen kan i denne opgave enten være studerende eller være i et job.

Derfor lav en base klasse der hedder Person, hvor afledte klasser Student og Employee arver fra Person. De skal også arve fra en interface som gøre man kan lave en Compare af to objekter. Dette bruges til sortering.

Programmet skal stille et **menu** til rådighed hvor brugeren kan få lov til at

·         indtaste en ny forekomst, opdatere en eksisterende,

·         delete eksisterende forekomst,

·         vise antal personer i listen, antal studerende og antal personer i job

·         vise personer med minimum og maximum alder eller med minimum og maximum løn.

·         give mulighed for at kunne finde en person udfra et telefon nummer.

·         håndtere exceptions og andre fejl på en passende måde

Lav exceptionklasse – måske en på klasse og en på kartotek og vis fejlene i en log måske på skærmen. Brug evt. samme log (med timestamp?) til at vise at en person bliver oprettet / Slettet.

Log Klasse? Singleton klasse. Måske et Interface! Så den bare tager mod en besked og det så er Log klassen der bestemmer hvordan data skal behandles.

·         generering af brugervenlige fejlmeddelelser,

·         mulighed for at gentaste fejlbehæftede felt.

Try catch på input med indikator på om et felt er forkert udfyldt. Try catch på hvert input, hvor catch kan vise tekst og rødt tegn ud fra hver boks.

Catch men udskriv funktionen?

·         kunne sortere listen, eksempelvis efter navn i faldende eller stigende orden eller begge muligheder.

* udskrive listen, enten studenter-listen eller employee-listen. (evt. valgmulighed om fil eller info på skærm)

Output file - Dan en fil med data. Få en Path (https://www.tutorialspoint.com/csharp/csharp\_file\_io.htm)

Du vælger selv om det skal være en console applikation eller en GUI, da her er der lagt vægt på at man skal kunne anvende klasser, interfaces, arving, liste, sortering, mm. Dvs. de mest vigtige begreber inden for OOP.

Det skal i hvert fald som minimum se sådan ud (hvis det er console, og hvis det er GUI så er det jer der bestemmer hvordan skal det se ud):

 Eksampel:

Operation Menu:

1. Tilføj eller Opdater ny person data. Applikationen skal tjekke om personen allerede findes eller ej. Brug telefonnummer som unik nøgle. Hvis den findes, opdater personen.   
  
2. Fjern en person fra listen (kom også med en brugervenlig meddelelse i tilfælde af om elementet findes eller ej)  
  
3. Vis antal personer I listen, antal studerende og antal personer i job  
  
4. Vis personer med Min og Max alder, eller løn  
  
5. Find en person  
  
6. Sorter listen og skriv ud på skærmen.  
  
7. Afslut programmet

apitel 1. Indledning ..........................................1

          1.1 Baggrund  ..........................................1

          1.2 Problemstilling ....................................1

          1.3 Deltagerliste ......................................1

  Kapitel 2. Analyse .............................................2

          2.1 Afgrænsning.........................................2

          2.2 Arugumentation for afgræsning.......................2

          2.3 Dataopsamling ......................................2

          2.4 Datastorage ........................................2

          2.5 Teknologi valg .....................................2

          2.6 Test strategi ......................................2

  Kapitel 3. Design ..............................................3

          3.1 Valg af klasser ....................................3

          3.2 Valg af lister .....................................3

  Kapitel 4. Programmering .......................................4

          4.1 Oprettelse af kunde ................................4

          4.2 Hent af kunde ......................................4

          4.3 Oprettelse af konti ................................4

          4.4 Indsættelse af beløb ...............................4

          4.5 ???? ...............................................4

  Kapitel 5. Test ................................................5

          5.1 Test scenarie 1 ....................................5

          5.2 Test Scenarie 2 ....................................6

          5.3 Test Scenarie 3 ....................................6

          5.4 Konklusion på test .................................7

  Kapitel 6. Konklustion .........................................8

          6.1 Indtryk, oplevelse og resultat .....................9

          6.2 Ting der ikke blev lavet ...........................9

          6.3 Forbedrings muligheder ............................10

          6.4 Udvidelses muligheder .............................10

1.1: Her fortælles om baggrund for opgaven.

Det er jo en eksamensopgave stillet i forbindelse med [Videregående programmering](https://studie.smartlearning.dk/mod/page/view.php?id=270817) ........

1.2: Hvad er det opgaven går ud på? mm.

Der skal laves en C# applikation til at oprette en eller flere liste-struktur der kan indeholde objekter af en eller flere klasse datastruktur.

Der skal laves en base klasse, der hedder Person, hvor afledte klasser Student og Employee arver fra Person. De skal også arve fra en interface som gøre man kan lave en Compare af to objekter. Dette bruges til sortering.

2.1: Analyse af problemstillingen.

Det er en rimelig afgrænset problemstilling, som burde kunne løses uden at der skal afgrænses noget. Da det dog er en læringsopgave og det ikke er en applikation, der skal bruges, vil en større exception handling og unit test begrænset, hvor det ikke længere giver læringsmæssigt værdi i forhold til deadlinie.

2.2:

Systemet fødes med data for at gøre testen lettere, men ellers indsamles der kun de data, der indtastes i selve applikationen.

2.4:

Pt. Bliver data gemt i lister, hvilket kunne ændres til en database, hvis behovet viste sig at opstå.

2.5:

C#, og VS

2.6:

Vi vælger både positive test scenarier, og negative test scenarier hvor vi indtaster forkert og noterer programmets virkemåde og retter hvis der er fejl mm.

3.1: Skitser de klasser det er hensigtsmæssigt at designe i applikationen

Yuml.me

%2F%2F Cool Class Diagram, [Person|\_name:string;\_age:int;\_adresse:string;\_postNumber:int;\_phoneNumber:int|CompareTo()]^-[Student|\_School:School],[Person]^-[Employee|\_job:string;\_salary:double]

[<<IComparable>>;CompareTo]^-.-[Person]

[Archive|Students:list;Employees:list;Persons:list;|CreateOrAlterStudent();CreateOrAlterEmployee();DeletePerson();ShowNumberOfPersons();ShowPersonsWithAge();ShowPersonsWithSalary();FindPersonWithPhoneNumber()]++-\*[Student]

[Archive]++-\*[Employee]

[Student]++-1[School]

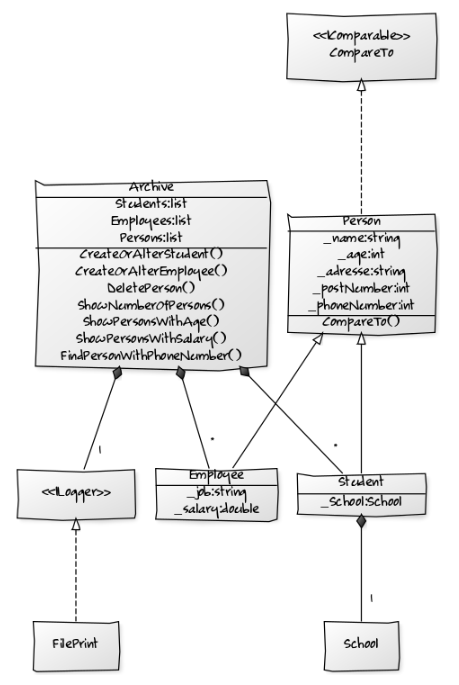
[Archive]++-1[<<ILogger>>]

[<<ILogger>>]^-.-[FilePrint]

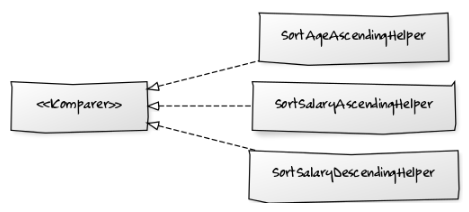
[<<IComparer>>]^-.-[SortSalaryDescendingHelper]

[<<IComparer>>]^-.-[SortSalaryAscendingHelper]

[<<IComparer>>]^-.-[SortAgeAscendingHelper]



Herudover er der 3 klasser der implementerer icomparer, samt en et unit test project og en enum klasse, der bliver brugt i FilePrint.



Design kan ses af ovenstående diagram, Hvor der laves en abstrakt person klasse, hvor der er to arvede klasser. Person klassen er lavet abstract da opgaveteksten går meget på at man skal være enten medarbejder eller studerende og det ikke er meningen at der skal være en instans af person klassen.

Persons implementere interfacet IComparable, som giver en default sortering (alder descending), som students og employees arver. Der er herudover også blevet lavet 3 classes som implementerer Icomparer<person>/<Employee>, så der kan laves en alternativ sortering.

Sort() har en overload som tager et Icompare interface, det er altså den klasse der implementer dette interface, der styrer sorteringen. Ved at give sort metoden en classe der sorter Employee efter salary desecending, er det den måde Sort funktionen sorter den pågældende liste af medarbejdere.

Jeg har valgt at Archive skal holde 2 lister. En med students og en med employees. Jeg tænkte oprindeligt, at have én liste og så lave en for løkke og caste til student/employee (AS operatoren) for så at validere om casting er succesfuldt, så jeg ved om det er en studerende eller ej – Men endte med at konkludere, at det blev for beregningstungt fremfor ”bare” at lave en type-safe liste, der kun holdte det ene eller det andet og så må man lave to løkker hvis man skal gennemgå begge lister.

Det eneste der ellers er lidt interessant i min opbygning er, at jeg har valgt at salary skal være nullable, det er sket dels for at lære denne feature, men også da det ofte sker at folk ikke ønsker at oplyse netop denne. Ved at lave den nullable, bliver feltet boxed til et objekt og det har en performance straf, men da det er et lille projekt og performace pt. Ikke er et issue accepteres dette.

I min Gui vil der i oprettelse være en radio button som bliver oversat til boolean værdi isStudent, så der bliver kaldt den korrekte CreateOrAlter metode – Jeg valgte 2 metoder, da de to konstruktors skulle bruge forskelligt input. Kiggede oprindeligt på at lave en metode, med optional parameters – men da jeg gerne vil lave salary nullable, da der er mange, der ikke vil opgive løn, var det lidt modstridende.

I mine sorteringsknapper lavede jeg også en ”Toggle”, så hvis man trykker på den samme knap igen, sker sorteringen modsat.

For at udvide brugen af interfaces har jeg valgt at uddelegere beskeder både i form af exceptions og generelle log-beskeder til et interface, hvor jeg har oprettet en klasse der implementerer interface, der udskriver til en fil. Det sker via dependecy injection i starten af formularen hvor objectet archive dannes (kunne også være lavet i Program.cs), så det vil være let at lave en ny klasse der eksempelvis udskriver til en printer eller lign. Og så skal archive bare have denne, når den oprettes i stedet.

4.1: Archive er den vigtigste overhovedet evt. employee

4.2: Employees

4.3 FilePrint?

.

5.1. Fortæl kort om testen, screenshot, samt resultatet og tilrettelse hvis der var fejl

Der er lavet et unitTest project, hvor der er lavet en unit test for hver af de hoved metoder der gennerer output. Der er lavet en test for hver udgang af metoden.

Disse er kørt for at sikre at back-end lever op til kravene.

Der er også lavet manuel test af front-end og for at gøre dette lettere, har jeg preudfyldt nogle studerende og medarbejdere, så man kan se min/max og antal uden at skulle oprette dem via brugergrænsefladen først.

5.2. Konklustion på test.

Det er en velfungerende applikation, dog kunne design og brugervenlighed forbedres.

6.1.

Alt i alt synes jeg at opgaven var god til at lære mere om interfaces og få dem ind under huden.

Opgaver

1. Blive færdig med at teste om alt det primæe virker (ret til så employee radio button åbner og lukker)
2. Lav kommentar til metoder mv.
3. tilføj de sidste del – vis alle personer (inkl. Sortering af disse) & udskriv listen knap – Tilføj funktionalitet
4. Lav exception handling – evt. der kalder print.
5. Lav Unit test på de X metode kald.
6. Lav print
7. Lav evt. lidt felt validering i felter – 4 cire på postnummer ect.

2.

Add.6. Udskriv - //interfaces.

Udskriv til en Fil:

Namespace xxxx

{

Public interface IPrint

{

void Print(string message)

}

}

//class der bruger Interface //dependency injection

Namespace xxxx

{

Public class Archive

{

Private readonly IPrint \_print;

Public Archive (IPrint print)

{

\_print = print;

}

Public Print (dsdfsfd ds fsd fdsf)

{

\_print.Print( resultat listen );

}

}

}

Public class FilePrint: IPrint

{

private readonly string \_path;

public FilePrint(string path)

{

\_path = path;

}

public void Print(string message)

{

Using (var streamWriter = new StreamWriter(\_path, true))

{

streamWriter.WriteLine(message);

}

}

}

//umiddelbart så skal lister laves til en string message.

If student

Archive.Print(”----Studerende fra resultat listen -----”)

Archive.Print(”{0} \t {1} \t {2} \t {3}”, format resultList[i].name, format resultList[i].name etc)

For (int i = 0; i < resultList.count; i++){

Archive.FilePrint.Print (”{0} \t {1} \t {2} \t {3}”, format resultList[i].name, format resultList[i].name etc.

If employee

Archive.Print (”----Medarbejdere fra resultat listen -----”)

For (int i = 0; i < resultList.count; i++){

Archive.Print (”{0} \t {1} \t {2} \t {3}”, format resultList[i].name, format resultList[i].name etc.

// I program.cs hvor der oprettes Archive skal der indsættes et interface //skriv kommentar.

Archvie archive = new Archive(new GuiLogger(), new FilePrint(“C:\\log.txt”));