# Øvelse 1: Terminologi

Nu skal du teste din forforståelse af dagens emne(r) med udgangspunkt i det asynkrone forløb.

Del teamet op i to mindre grupper, og brug **Møde på midten** (beskrivelse følger i punktform) i hver gruppe til at diskutere begreberne ”persistens”, ”StreamWriter”, ”StreamReader”, ”DateTime” og ”string Split-metoden”. Husk, at alle i gruppen skal have taletid:

1. Hvert enkelt medlem i gruppen skriver sine overvejelser om begreberne ned (2 minutter).
2. Hver gruppe starter **Ordet rundt**, idet det første gruppemedlem forklarer sine overvejelser   
   (2 minutter). Hele gruppen diskuterer bidraget (2 minutter). Hvis gruppen enes om, at det er en god forklaring, skriver det første gruppemedlem sit bidrag ind i gruppens dokument.
3. Det næste gruppemedlem forklarer sine overvejelser osv. Fortsæt med **Ordet rundt** indtil alle 4 gruppemedlemmer har bidraget, eller tiden er gået.

*Tidsramme: 20 minutter*

# Øvelse 2: Klargøring

Udfør følgende:

* Opret en ny konsolapplikation med projektnavnet ”Persistens”
* Opret et testprojekt med navnet ”PersistensTest”. Sørg for, at testprojektet har en reference til konsolprojektet samt at testen også kender til konsolprojektets namespace.
* Indsæt den unit-test angivet i filen *Persistens.UnitTest.docx*.

Udfør disse trin nu, inden du fortsætter.

Dagens opgave handler om at persistere data. Første trin er at etablere data, du gerne vil gemme til og hente fra en tekstfil. I den næste øvelse skal du derfor oprette en Person-klasse som datagrundlag, hvor du repeterer det, du har lært i forrige uge om properties og constructors. Derefter følger en øvelse, hvor du skal implementere kode til at gemme data i en tekstfil, og hente det gemte ind i programmet igen.

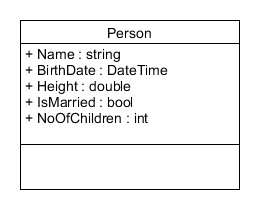
**Benyt parprogrammering til de følgende øvelser**, så ingen sidder alene.

# Øvelse 3: Person-klassen

Du skal starte med at definere en Person-klasse, som repræsenterer en person med forskellige informationer, derefter udbygger du klassen med en constructor og en metode.

## Øvelse 3.1: Klasse med properties

Du skal tage udgangspunkt i følgende designklasse for en Person:



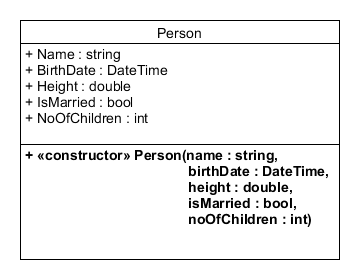
Bemærk, at den valgte navnekonvention kræver, at public attributter (med + foran), der starter med et stort bogstav (PascalCase), skal implementeres som properties i den implementerede C#-klasse. De private felter, som er ’backing fields’ for hver property, er ikke angivet i designklassen, men skal implementeres i C#-klassen med et lille startbogstav (camelCase).

Udfør følgende:

* Tilføj en ny C#-klasse med navnet ”Person” til projektet
* Implementér alle angivne attributter i designklassen som C# properties og deres tilhørende bagvedliggende private felter (backing fields) i Person-klassen, hver med følgende betydning:
  + *Name* (string): angiver personens fulde navn
  + *BirthDate* (DateTime): angiver personens fødselsdato
  + *Height* (double): angiver personens højde som et reelt tal
  + *IsMarried* (bool): angiver om personen er gift (true/false)
  + *NoOfChildren* (int): angiver hvor mange børn personen har

## Øvelse 3.2: Constructor

Designklassen udvides med en constructor (angivet i fed skrift) som vist forneden:

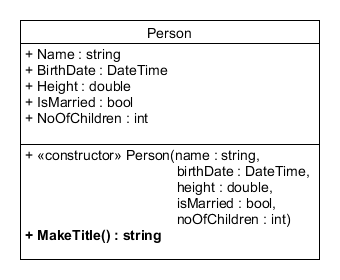


Udfør følgende:

* Definer den nye constructor i klassen med den angivne header, som initialiserer alle objektets properties med parametrene: *name*, *birthDate*, *height*, *isMarried* og *noOfChildren*

## Øvelse 3.3: MakeTitle()-metoden

Tilsidst skal du tilføje en ny metode, som vist forneden:



MakeTitle()-metoden skal returnerer en tekststreng, som består af værdien af hver af klassens properties adskilt med et semikolon (;). Der må ikke være ekstra mellemrum før og efter et semikolon. Et eksempel på en returneret tekststreng kunne være:

* ”Anders And;01-05-1962 00:00:00;175,5;True;2”

**Hint**: husk at variable med datatyperne DateTime, double, bool og int selv kan konvertere til en tekststreng. Det har du allerede set i Console.WriteLine()-metoden.

Udfør følgende:

* Implementér *MakeTitle()*-metoden i Person-klassen som angivet foroven.

## Øvelse 3.4: Test din nye klasse

Det er nu tid til at teste, om din implementering af Person-klassen er korrekt.

Udfør følgende:

* Åbn Test Explorer, og kør unit-testen. Det vil kun være den første testmetode *CheckPersonConstructor()*, som er aktiv til at starte med.

Bemærk, at tekstrepræsentationen af DateTime og decimaltal følger det sprog, som du har valgt i din installation af Windows (eller et andet operativsystem), så hvis Windows-installation ikke er dansk, men f.eks. engelsk, vil tekstformatet af en dato og også et decimaltal være anderledes end det angivne i testen. Man kan angive formateringen i en ToString()-angivelse for en DateTime-variabel. Hvis din Windows-installation er på engelsk, da kan du tilføje følgende til din BirthDay-property:

BirthDay.ToString("dd-MM-yyyy HH':'mm':'ss")

Hvis du har lyst, kan du læse mere om ’format specifiers’ for DateTime her: [C# DateTime Format](https://www.c-sharpcorner.com/blogs/date-and-time-format-in-c-sharp-programming1)

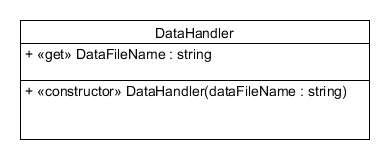
Prøv om du selv kan finde ’format specifiers’ for double, hvis du har brug for det.

# Øvelse 4: Persistens

I denne øvelse skal du sørge for, at data fra et Person-objekt kan skrives til en tekstfil, og data kan hentes fra tekstfilen til et Person-objekt igen. Denne funktion skal håndteres via din egen DataHandler-klasse, en særskilt C#-klasse du nu skal implementere.

## Øvelse 4.1: Property og constructor

Designklassen for DataHandler er angivet forneden:



Navnekonvention: Attributter med ’+’ og stort startbogstav skal implementeres som en C#-property. Feltet bag denne property (backing field) er ikke angivet i designklasse, men skal også defineres med samme navn, men i camelCase (lille startbogstav).

Designklassen indeholder en attribut og en constructor (til at starte med):

* *DataFileName* (string): en attribut der angiver filnavnet på den tekstfil, der benyttes til at gemme data i. Attributten kan kun læses og ikke sættes direkte, hvilket er angivet af stereotypen ’get’. Ifølge navnekonventionen skal attributten således implementeres som en C#-property i klassen og kun med en getter. Bemærk, at attributten kun kan tildeles en værdi via klassens constructor (se også forneden).
* *DataHandler(dataFileName : string)*: en constructor som kaldes ved instantiering af et DataHandler-objekt. Parameteren *dataFileName* angiver filnavnet på den tekstfil, data skal gemmes i, og tildeles til *DataFileName*-property i constructor’en . Da en constructor kun kaldes en gang for et objekt, kan *DataFileName*-property’en ikke ændres efterfølgende ude fra klassen selv.

Udfør følgende:

* Tilføj en C#-klasse med navnet *DataHandler* til projektet
* Implementér en *DataFileName*-property kun med en getter
* Implementér en constructor til DataHandler-klassen med en parameter som bruges til at initialisere DataFileName-property’en med

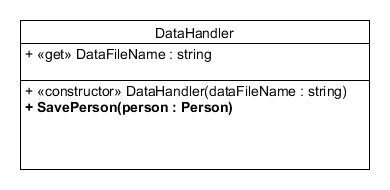
Udfør testmetoden *CheckDataHandlerConstructor()* i unit-testen:

* Fjern udkommenteringen (//) foran ’[TestMethod]’ en linje over testmetoden *CheckDataHandlerConstructor()*
* Kør testen, og tilret om nødvendigt

Udfør disse trin nu, inden du fortsætter.

## Øvelse 4.2: Skriv til tekstfil

Designklassen for DataHandler opdateres nu til:



Designklassen udvides nu med en ekstra metode (vist i fed skrift):

* *SavePerson(person : Person)*: en metode der gemmer data for det i parameteren angivne Person-objekt i en tekstfil med filnavnet angivet i *DataFileName*-property’en.

Udfør følgende:

* Implementér *SavePerson()*-metoden med brug af StreamWriter-klassen og Person-klassens MakeTitle()-metode.
* Tilføj følgende kode til din Main()-metode i Program-klassen:

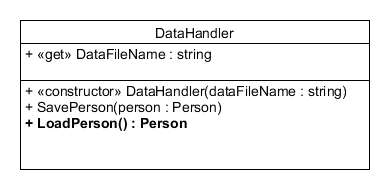


* Kør programmet, og inspicér den genererede tekstfil ’Data.txt’ for, om data er indsat korrekt.

For at inspicere ”Data.txt” skal du først finde ud af, hvilken mappe dit program faktisk har gemt tekstfilen i. Når man har angivet et filnavn som ovenfor uden en foranstillet sti og benytter StreamWriter-klassen, så placeres tekstfilen i den mappe, hvor programmets exe-fil, dvs. den eksekverbare fil for programmet, også er placeret. Denne mappe kan du finde ved at højreklikke på konsolprojektet i Solution Explorer, dvs. på ’Persistens’-projektet, og vælge punktet ’Open Folder in File Explorer’. I File Explorer vælger du så mappen ’bin’, dernæst ’Debug’ og tilsidst ’net6.0’ (hvis du har oprettet et .Net 6.0-projekt, ellers en anden tilsvarende mappe). I denne undermappe vil du finde din ’Data.txt’-fil, som du kan inspicere ved at åbne den i Notepad eller en lignende teksteditor.

## Øvelse 4.2: Læs fra tekstfil

Vi er ikke helt færdige endnu, så designklassen for DataHandler opdateres igen:



Designklassen udvides nu med en ekstra metode (igen vist i fed skrift):

* *LoadPerson() : Person*: en metode der henter data fra en tekstfil med filnavnet angivet i *DataFileName* og derefter opretter og returnerer et tilsvarende Person-objekt. Der skal selvfølgelig eksistere en fil med det angivne filnavn for, at den kan åbnes og læses.

Udfør følgende:

* Implementér *LoadPerson()*-metoden med brug af StreamReader-klassen. Når du har læst en linje fra tekstfilen, så har du reelt hentet en tekststreng, som du skal opsplitte i enkelte værdier og indlæse disse i et nyt Person-objekt. Du kan vælge at bruge Split()-metoden samt Parse()-metoder for de respektive datatyper, der indgår i Person-klassens felter/properties, til denne operation.  
  Overvej nøje trinene, før du koder metoden.

Udfør testmetoden *CheckDataHandlerSaveAndLoad()* i unit-testen:

* Fjern udkommenteringen (//) foran ’[TestMethod]’ en linje over testmetoden *CheckDataHandlerSaveAndLoad ()*
* Kør testen, og tilret om nødvendigt.  
  Debug din testkode trin for trin, så du forstår den bedst muligt.

# Øvelse 5: Review-spørgsmål

* Hvilket .Net-bibliotek (namespace) skal man benytte sig af for at kunne bruge StreamWriter/StreamReader?
* Hvorfor er det vigtigt at lukke ens StreamReader/StreamWriter?
* Hvad kræves det, før en klasse kan ses i et andet namespace?
  + Hint: Access modifier
* Hvad kan du bruge DateTime-datatypen til?

# Øvelse 6: Terminologi

Start med at teste din forforståelse af dagens emne(r) med udgangspunkt i dagens forberedelse.

Del teamet op i to mindre grupper, og brug **Møde på midten** i hver gruppe til at diskutere begreberne ”exception handling” og ”constructor overloading”. Husk, at alle i gruppen skal have taletid.

*Tidsramme: 20 minutter*

# Øvelse 7: Flere personer

**Hvis du ikke fik lavet øvelse 3 og 4 færdig inkl. testet din løsning med den udleverede unit test, så sørg for at gøre dette, inden du går videre.**

Du skal nu videreudvikle den løsning, du implementerede i forrige opgave, til at persistere mange personer og ikke bare én enkelt person. Dvs. du fortsætter med de to projekter, du oprettede: din ”Persistens”-konsolapplikation samt testprojektet ”PersistensTest”.

**Benyt parprogrammering til de følgende øvelser.**

## Øvelse 7.1: Overvej algoritmen

Designklasserne fra forrige opgave er samlet i følgende DCD (designklassediagram):

Et billede, der indeholder tekst, skærmbillede, Font/skrifttype, linje/række

Automatisk genereret beskrivelse

DCD’et viser også en ny udvidelse af DataHandler-klassen, hvor to nye operationer er tilføjet (fed skrift):

* *SavePersons(persons : Person[])*: gemmer den samling af personer, der er angivet i parameter’en *persons*, i DataHandler’ens tekstfil, hvis navn angivet i *DataFileName*-property’en.
* *LoadPersons() : Persons[]*: returnerer den samling af personer, der er lagret i DataHandler’ens tekstfil, hvis navn angivet i *DataFileName*-property’en.

Disse to operationer svarer til *SavePerson()* og *LoadPerson()*, men skal nu persistere en samling af personer i stedet for bare én enkelt person.

Udfør følgende:

* Overvej først sammen med din kodepartner, hvordan algoritmen til *SavePersons()* skal opbygges. Beskrive algoritmen på papir eller whiteboard, så du er klar over, hvad du vil, inden du begynder at kode i C#.
* Overvej på samme måde, hvordan operationen *LoadPersons()* skal opbygges algoritmisk.

**Udfør disse trin nu, inden du fortsætter.**

## Øvelse 7.2: Implementér i C#

Med dine algoritmiske overvejelser i hånden, udfør nu følgende:

* Implementér først operationen *SavePersons()* som en metode i C#, hvor Person-samlingen i *persons*-parameteren skal implementeres som et array i C#. Hvert Person-objekt i *persons* skal gemmes i en separat linje i tekstfilen.
* Implementér dernæst operationen *LoadPersons()* i C#.

## Øvelse 7.3: Test din implementering

Du skal nu teste din nye implementering via nye testmetoder i din unit-test.

Udfør følgende:

* Åbn din eksisterende unit-test fil
* Indsæt alle testmetoderne (også de udkommenterede) angivet i filen *Persistens2.UnitTest.docx* (hentes via Git materiale-folderen – benyt git pull)
* Gennemse den nye testmetode *CheckDataHandlerSaveAndLoadManyPersons()*, så du forstår, hvad det er, testmetoden egentlig tester.
* Åbn Test Explorer, og kør unit-testen.
* Tilret, din kode om nødvendigt. Test igen, til alle tests er grønne.

# Øvelse 8: Finpudsning af Person-klassen

Person-klassen skal finpudset lidt, så den er mere brugervenlig og håndterer ugyldige data bedre.

## Øvelse 8.1: Constructor overloading

For at gøre Person-klassen lidt mere brugervenlig, så skal der laves en ekstra constructor, hvor man ikke behøver at angive antal børn (NoOfChildren), men ellers alle de andre 4 parametre. Når antal børn ikke er angivet, **så antages antallet at være nul som default**.

Udfør følgende:

* Implementér en ekstra (overloaded) constructor til Person-klassen uden en *noOfChildren*-parameter, men med de 4 andre parametre. Brug *this*-notationen for denne nye constructor til at kalde den første constructor med alle 5 parametre, hvor noOfChildren sættes til default-værdien (se foroven).

## Øvelse 8.2: Gør Person-klassen mere sikker

Det er muligt at angive forkerte oplysninger, når du enten instantierer Person-klassen eller efterfølgende ændrer en property i et eksisterende Person-objekt.

Det skal du nu håndtere ved at sikre, at følgende krav er opfyldt, når du instantierer eller ændrer et Person-objekt:

* En person skal have et navn med mindst en karakter. Dvs. en tom tekststreng er ikke acceptabel som navn.
* Højden på en person skal være større end 0.
* Fødselsdatoen for en person må ikke være før 1. januar 1900.
* En person må ikke have et negativt antal børn.

Udfør følgende:

* Tilret din kode for Person-klassen, så ovennævnte krav er opfyldt.   
  Hint: det kan være en god ide at se på dine property’s.

Hvis du detekterer en fejl, da opret og ’kast’ en exception via sætningen:

* throw new Exception(”Der er opstået en fejl … eller hvad der nu er relevant at skrive her”);

Detaljerne omkring at kaste sine egne exceptions bliver dækket i forberedelsen om kort tid. Lige nu skal du bare bruge ovenstående sætning. Du må gerne ændre fejlmeddelelsen, så den passer til konteksten.

## Øvelse 8.3: Test Person-klassen

Tiden er inde til at teste din finpudsning af Person-klassen

Udfør følgende:

* Fjern udkommenteringen af de resterende testmetoder i det udleverede.
* Gennemse de forskellige testmetoder, så du forstår, hvad det er, de tester.
* Kør unit-testen.

## Tilret, din kode om nødvendigt, indtil alle tests er grønne.

# Bonusøvelse 9: Multiple klasser

Din løsning håndterer kun én type objekter, nemlig objekter fra Person-klassen. I en typisk løsning vil der være adskillige forskellige objekttyper, som skal persisteres, og oftest med indbyrdes sammenhæng.

## Bonusøvelse 9.1: Overvej andre klasser i din løsning

Udfør følgende:

* Overvej hvilke andre typer klasser kunne være relevante for Person-klassen i din løsning. Det kunne f.eks. være en Team-klasse, som kan organisere personer i teams, eller helt andre typer klasser.
* Udvælg en eller to af disse klasser, og fastlæg, hvilke attributter og operationer kunne være relevante for dem samt hvilke(n) association, der er til Person-klassen.

## Bonusøvelse 9.2: Implementér flere klasser

Udfør følgende:

* Implementér en (eller to) af dine fundne klasser i C# med deres properties og metoder.

## Bonusøvelse 9.3: Persistens af nye klasser

Udfør følgende:

* Nu hvor du har mere end én klasse at persistere, overvej hvordan du vil sørge for at persistere dem alle:
  + Skal objekter fra hver klasser persisteres i hver sin tekstfil?
  + Eller vil du have, at de alle persisteres i samme tekstfil? Hvis det sidste er dit valg, hvordan kan du så skelne mellem forskellige typer objekter i én tekstfil?

Det er op til dig, hvilken løsning du vælger, men sørg for først at overveje, hvordan du vil repræsentere forskellige objekter i en tekstfil samt den algoritmiske side af løsningen, før du begynder at implementere den.

* Implementér din løsning, og afprøv, om den virker.