# Persistence (da: Persistens)

|  |  |
| --- | --- |
| **Læringsmål** | Du kan:   * **1Pf1:** anvende centrale metoder til at specificere og konstruere algoritmer [...] * **1Pf2:** anvende centrale faciliteter i programmeringssproget til realisering af algoritmer […] * **1Pf3**: anvende et i professionen udbredt, integreret udviklingsværktøj, herunder versionsstyringssystem […] til at designe og konstruere praksisnære applikationer […] * **1Pk3**: i en struktureret sammenhæng tilegne sig ny viden, færdigheder og kompetencer inden for programmeringssprog, udviklingsværktøjer, programmeringsteknikker og programdesign |
| **Forventet læringsudbytte [SOLO]** | Programmeringssporet:   * [Unistrukturel] Du kan genkende enkelte kendetegn ved **datatyper** - int, string, bool, double, char, array, DateTime * [Unistrukturel] Du kan genkende enkelte kendetegn ved **C#** **klasse** – klasse, felt (en: field), metode, constructor, property (get, set), access modifiers (public, private), access keywords (this) * [Unistrukturel] Du kan nævne enkelte kendetegn ved **Persistens (tekstfiler)** –streams (StreamReader, StreamWriter), tekstfil (læsning og skrivning)   Færdighedssporet:   * [Unistrukturel] Du kan genkende enkelte kendetegn ved **testing** |
| **Din forberedelse** | Programmeringssporet:   * [Working with DateTime](https://channel9.msdn.com/Series/C-Fundamentals-for-Absolute-Beginners/14) (Bob Tabor, video: 16:22) * [Date and time in C#](https://zetcode.com/csharp/datetime/) (dækker både DateTime og TimeSpan) Læs afsnittene:   + Til og med: ’C# add and subtract DateTime’ (i alt 5 afsnit)   + Og herefter: ’C# TimeSpan’, ’C# format time’ og ’C# parse time’ * [C# Split String Examples](https://www.dotnetperls.com/split) (læs til og med afsnittet ’Text files’, men spring ’Regex.Split’ over) * [YB] 3.6 (s. 73–78) Using Files * [‘How to read from and write to a text file’](https://support.microsoft.com/en-us/help/816149/how-to-read-from-and-write-to-a-text-file-by-using-visual-c) |

Dagens opgave er en introduktion til persistens, dvs. helt kort: det at gemme data fra et program, så samme data senere kan hentes frem igen i programmet. Du vil også i denne opgave for første gang stifte bekendtskab til datatypen DateTime, som samlet repræsenterer både tid og dato, samt TimeSpan, som repræsenterer en tidsperiode. I opgaven skal du gemme data fra dit program i en tekstfil og efterfølgende læse disse data ind i programmet igen. Dette kræver kontakt til det underliggende operativsystem (Windows eller evt. MacOS), specielt filsystemet.

# Dagens ord: Øvelse gør mester

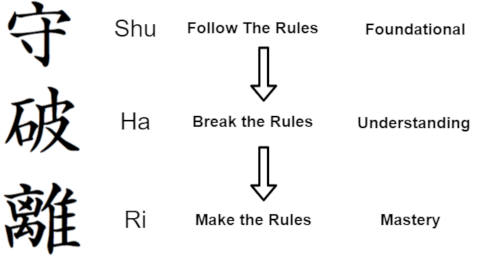
Det er vigtigt, at du som studerende (og senere i livet) benytter de antal timer, som der skal til, når du forsøger at opnå en færdighed. Igen er det vigtigt at nævne, at størrelsen af dit hjemmeprojekt betyder mindre, end at du får øvet dig.

En agil læringsteknik, som kan benyttes som *inspiration* (der kommer fra Japansk kampsport, Aikido), og som også komplimenterer vores læringstaksonomi SOLO, er Shu-ha-ri.

**Shu:** Du skal fokusere på, hvordan du udfører arbejdet/løser en opgave, for eksempel skal du fokusere på, hvordan du skriver en metode, laver et objekt, benytter et loop, nedbryder et problem. Det er meget vigtigt at have en at sparre med. Feedback er nødvendigt.

**Ha:** Du som studerende begynder at fokuserer på at opnå en forståelse for den underliggende teori og de principper, der ligger til grund for det, du gør. Refleksion, tiltag samt feedback er nødvendigt. Du skal opsøge flere end blot *én* medstuderende eller *én* underviser.

**Ri:** Du som studerende har ikke fokus på at lære gennem andre længere, men igennem det du gør, din praksis. Du reflekterer over egne erfaringer og tilpasser løbende din tilgang i henhold til de forudsætninger, der præsenteres for dig. Refleksion og tiltag er nødvendigt her.



# Øvelse 1: Terminologi

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

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 *Ex11-Persistens.UnitTest.docx*, som du kan hente i Git materiale-folderen (benyt git pull).

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.

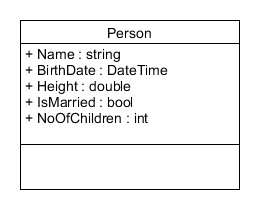
**Benyt parprogrammering til de følgende øvelser**, så ingen sidder alene.

# 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:



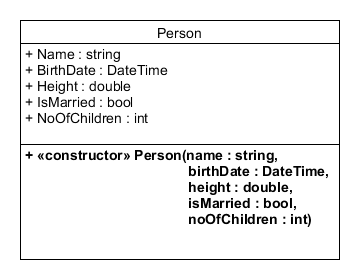
Bemærk, at den valgte navnekonvention kræver, at public attributter (med + foran), der starter med et stort bogstav (PascalCase), skal implementeres som properties i den implementerede C#-klasse. De private felter, som er ’backing fields’ for hver property, er ikke angivet i designklassen, men skal implementeres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

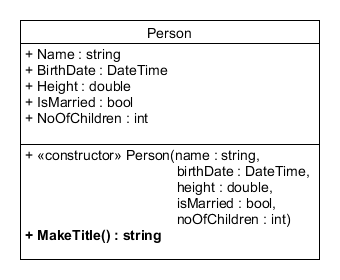


Udfør følgende:

* 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:

* ”Jan Brown;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)

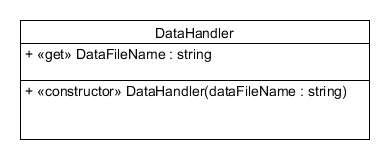
Prøv om du selv kan finde ’format specifiers’ for double, hvis du har brug for det.

# 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

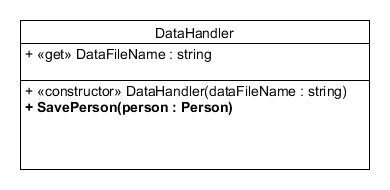
Udfør testmetoden *CheckDataHandlerConstructor()* i unit-testen:

* 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:

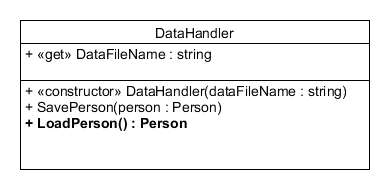


* Kør programmet, og inspicér den genererede tekstfil ’Data.txt’ for, om data er indsat korrekt.

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?