# Persistence 2

|  |  |
| --- | --- |
| **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 simple C# **datatyper** - int, string, bool, double, char, array, DateTime * [Unistrukturel] Du kan genkende enkelte kendetegn ved **C#** **klasse (objekt)** – C# klasse, felt (en: field), metode, constructor, property (get, set), access modifiers (public, private), access keywords (this) * [Unistrukturel] Du kan nævne enkelte kendetegn ved **File I/O** – streams (StreamReader, StreamWriter), tekstfil (læsning og skrivning)   Færdighedssporet:   * [Unistrukturel] Du kan genkende enkelte kendetegn ved **testing** –unit test, test class, test method, assertion (Assert-klasse), test class attribute ([TestClass], [TestInitialize], [TestMethod]), Test Explorer |
| **Din forberedelse** | Programmeringssporet:   * [Exception handling](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/exceptions/exception-handling) (try, catch & finally) * [YB] afsnit 4.7.3 ”*Overloading Constructors*” til og med 4.7.4 ”*Constructor Management*”, side 104-107. * Læs om [C# Constructor Overloading](https://www.geeksforgeeks.org/c-sharp-constructor-overloading/) * Genlæs/-se evt. forberedelsen til Ex11-Persistence samt videoen ”Indkapsling – Properties” fra opgave Ex10-Properties |

**Sørg for at gøre øvelse 3 (Person-klasse) og 4 (Persistens) i opgaven Ex11-Persistens færdig som forberedelse, da du arbejder videre med disse øvelser i opgaven her.**

Du skal fortsætte med at arbejde med persistens i denne opgave og også stifte bekendtskab med begrebet overloading i forbindelse med constructors. Derudover repeterer du også en del tidligere emner.

# Øvelse 1: 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 2: Flere personer

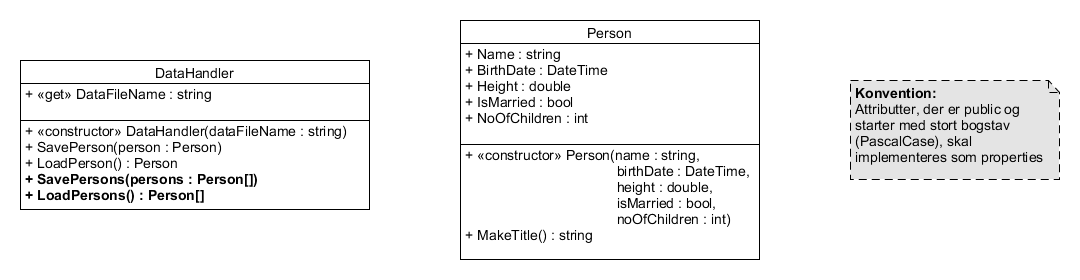
**Hvis du ikke fik lavet øvelse 3 og 4 i Ex11-Persistens færdig inkl. testet din løsning med den udleverede unit test, så sørg for at gøre dette, inden du går videre.**

Du skal nu videreudvikle den løsning, du implementerede i opgave Ex11-Persistens, 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 2.1: Overvej algoritmen

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



DCD’en 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. Benyt teknikkerne i Computational Thinking, og også gerne pseudokode, til at 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 2.2: 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 2.3: 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 *Ex12-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 3: Finpudsning af Person-klassen

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

## Øvelse 3.1: 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’en med alle 5 parametre, hvor noOfChildren sættes til default-værdien (se foroven).

## Øvelse 3.2: 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 fejlmeddelelsen, så den passer til konteksten.

## Øvelse 3.3: 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 4: 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. Dette vil f.eks. være tilfældet med de klasser, du og dit team har valgt at lade indgå i jeres Hydac-projekt.

## Bonusøvelse 4.1: 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 4.2: Implementér flere klasser

Udfør følgende:

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

## Bonusøvelse 4.3: 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.