# C# Datahåndtering

|  |  |
| --- | --- |
| **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 |
| **Forventede  produkter** | * Implementering af programmeringsøvelserne |
| **Forventet læringsudbytte [SOLO]** | Designsporet:   * [Unistrukturel] Du kan identificere enkelte kendetegn ved **Computational Thinking** – dekomponering, mønstergenkendelse (en: pattern recognition) * [Unistrukturel] Du kan genkende enkelte kendetegn ved **strukturmodellering** – konceptuel/ software design klasse, klassenavn, attribut (navn, datatype)   Programmeringssporet:   * [Unistrukturel] Du kan genkende enkelte kendetegn i en C# **programstruktur** – sætning (en: statement), kodeblok (en: code block) * [Unistrukturel] Du kan genkende enkelte kendetegn ved C# **datatyper** – int, string, bool, double, char * [Unistrukturel] Du kan nævne enkelte kendetegn i **datahåndtering** – variabel, virkefelt (en: scope), udtryk (en: expression) og operatorer i disse samt tildeling (en: assignment) og tilhørende tildelings­operatorer   Færdighedssporet:   * [Unistrukturel] Du kan genkende enkelte kendetegn ved **versionsstyring** (en: version control) – GitLab login, Git Bash, Git-kommandoer: init, status, add, commit og push |
| **Din forberedelse** | Designsporet:   * [Computational Thinking: Pattern Recognition](https://www.youtube.com/watch?v=SixLnIDV1yY) (video: kun de første 7:29) * [Pattern Recognition (BBC)](https://www.bbc.co.uk/bitesize/guides/zxxbgk7/revision/1) (3 sider + Test og Score)   Programmeringssporet:   * [An introduction to object and classes](https://www.youtube.com/watch?v=tWIe9E4SWQo) (video: 8:45) * [YB] 2.1 - 2.2 (s. 14-33) * [Operators, Expression, Statements](https://channel9.msdn.com/Series/CSharp-Fundamentals-for-Absolute-Beginners/Operators-Expressions-and-Statements) (video: 15:22) * [Floating points](https://csharp.net-tutorials.com/data-types/floating-points/) * [C# Character Data Type](http://www.blackwasp.co.uk/CSharpCharacterDataType.aspx) (se bort fra nullable character; det kommer vi til) * [String](https://www.youtube.com/watch?v=Gw9kLUhV0dQ) (video: 2:18) * [Working With Strings](https://code-maze.com/csharp-basics-string-methods/)   Færdighedssporet:   * [Introduction to Git](https://www.notion.so/Introduction-to-Git-ac396a0697704709a12b6a0e545db049) (til og med punkt 4.4) * [Learn Git in 15 minutes](https://www.youtube.com/watch?v=USjZcfj8yxE) (video: 00:00 – 10:45)   + Fokus på git kommandoer: config, init, status, add, commit, push, log, staging og repository |

Du skal i denne opgave se videre på datahåndtering med udtryk og operatorer, introduceres til to nye datatyper double og char samt klasser. Derudover skal du se på mønstergenkendelse i CT og begynde med Git. **Sørg for at gøre øvelse 3 og 4 i Ex02-Conditionals færdig som forberedelse, da de vil indgå i denne opgave.**

# Dagens ord: Mønstre og objekter

I dagligdagen oplever vi mange slags mønstre, både bevidst og ubevidst, og oftest ser vi sammenhænge mellem forskellige objekter, vi ser hver dag, f.eks. at forskellige objekter også kan have fælles egenskaber. Selvom to bygninger sjældent er helt ens, så vil der altid være visse fællestræk, som kendetegner en bygning (tag, døre, vinduer, etc.). Det samme gælder for en bil, en cykel, osv. Selve opgaven at finde de træk, der er fælles, og dem som er forskellige, er netop tanken bag mønstergenkendelse i Computational Thinking. Dette skal du arbejde med i denne opgave samt med et par ekstra datatyper, beregninger og GIT.

# Øvelse 1: Computational Thinking – Mønstergenkendelse

Mønstergenkendelse er en helt central disciplin til at finde fællesnævnere mellem forskellige sammenhænge og til at forsimple forløb, der ellers kan være meget komplekse; procesoptimering indeholder primært mønstergenkendelse omkring arbejdsgange.

## Øvelse 1.1: Terminologi

Del teamet i to grupper, og brug CL-strukturen **Møde på midten** til at reflektere over begrebet ”mønstergenkendelse”.

1. Reflektér over begrebet individuelt (1-2 min.)
2. Hvert gruppemedlem præsenterer dernæst resultatet af deres refleksion (1-2 min.)
   * Gruppen diskuterer det præsenterede (1-2 min.)
   * Bliv ved, indtil alle gruppemedlemmer er blevet hørt

*Tidsramme: 15 minutter*

## Øvelse 1.2: Mønstre i dagligdagen

Find mønstre i dagligdagen:

* Brug 2 min. individuelt til at liste mønstre, som du har observeret i dagligdagen.
* Brug 8 min. (1 min. hver) til at præsentere jeres liste hurtigt for de andre i teamet.

*Tidsramme: 10 minutter*

## Øvelse 1.3: Mønster i kode

Hvis du sammenligner dine løsninger til øvelse 3.2 henholdsvis 3.3 fra forrige opgave (Ex02-Conditionals), så handler de om to vidt forskellige løsninger (3.2: navn/alder, 3.3: menuvalg).

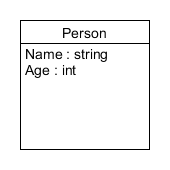
* Brug 2 min. individuelt til at kigge på dine to løsninger, og overvej om, der er et mønster i selve koden, som er ens for begge løsninger? Hvis ja, hvad er dette mønster?
* Benyt en hurtig **Ordet rundt** (husk at høre alle 1 min.), og præsentér dine fund for de andre i teamet.

*Tidsramme: 10 minutter*

# Øvelse 2: Konceptuel klasse til software designklasse

Du har i forrige opgave Ex02-Conditionals inspiceret og fundet kendetegn for en konceptuel Person-klasse, som beskriver en klasse af objekter fra det virkelige liv. Du ønsker nu at angive Person-klassen på et mere konkret ”software”-niveau, nemlig som en design-klasse, og ikke kun på konceptuelt niveau.

Dette gør du (i første omgang) ved at angive datatypen for hver af de enkelte attributter Name og Age (tidligere Navn og Alder), de to egenskaber ved Person-klassen.



Dette skift af niveau fra konceptuel klasse til software designklasse indikerer også et skift for modtageren af beskrivelsen, nemlig fra f.eks. en kunde uden forståelse for udvikling af et it-system til en softwareudvikler, der faktisk ved, hvad int og string betyder og har styr på de engelske fagtermer. Du har sikkert også bemærket sprogskiftet fra dansk til engelsk, hvilket typisk sker, fordi målgruppen nu er udviklere. Så selv om de to klasser ligner hinanden og tydeligt har en sammenhæng, så har de vidt forskellige målgrupper og anvendelsesområde. Som du allerede ved nu fra systemudvikling, så bruges konceptuelle klasser i domænemodellen.

## Øvelse 2.1: Omdan Person-klasse til software-klasse

Med udgangspunkt i alle de kendetegn for Person-klassen, du fandt i øvelse 4 i Ex02-Conditionals, omdan nu din egenudviklede Person-klasse til en tilsvarende software-designklasse ved at angive datatypen for hver af de attributter (egenskaber) du har fundet, dvs. udover Name og Age. Dvs. benyt engelsk navngivning. Lav gerne et diagram med velegnet tegneværktøj (forslag: UMLet, PowerPoint, Visio, mm.).

*Tidsramme: 10 minutter*

## Øvelse 2.2: Beskriv flere klasser

Med Computational Thinking i baghovedet så brug dekomponering (nedbrydning) og mønstergenkendelse til at beskrive to andre klasser, som du kender fra din hverdag.

Udfør følgende:

* Dan en konceptuel klasse for både en bil (en Bil-klasse) og et (generisk) produkt (en Produkt-klasse) med alle de generelle egenskaber, du tænker kan være relevante.
* Dan nu ud fra disse to klasser to tilsvarende software-designklasser, Car-klassen og Product-klassen, hvor du nu anfører datatypen for hver attribut, du har fundet. Bemærk et skift her i sprog (fra dansk til engelsk). Dette berøres senere, men giv gerne et gæt på, hvorfor det kunne give mening at ændre sproget.

*Tidsramme: 15 minutter*

# Øvelse 3: Beregninger med int og double

Du skal i denne øvelse arbejde primært med de to datatyper int og double, udføre forskellige typer beregninger og blive mere fortrolig med variabler samt indlæsning fra og udskrivning til konsolvinduet.

Udfør følgende:

* Opret en helt ny konsolapplikation til denne øvelse, hvor du skal implementere de angivne beregninger i følgende øvelser, hvor du (til at starte med) udelukkende anvender int-typen til at gemme data med. **Al kode skrives inden i Main()-funktionen, også erklæring af evt. lokale variable.** Brug Console.ReadLine() og Console.WriteLine() til at kommunikere med brugeren.

## Øvelse 3.1: Rektanglets areal

Beregn arealet af et rektangel ud fra formlen A = højde \* bredde:

* Indlæs højde på rektanglet
* Indlæs bredde på rektanglet
* Beregn og udskriv arealet

Afprøv din kode, og beregn forskellige arealer.

## Øvelse 3.2: Hældning af linjestykke

Beregn hældning af linjestykke med startpunkt i (x1, y1) og slutpunkt i (x2, y2) ud fra formlen h = :

* Indlæs startpunktets x-koordinat x1
* Indlæs startpunktets y-koordinat y1
* Indlæs slutpunktets x-koordinat x2
* Indlæs slutpunktets y-koordinat y2
* Beregn og udskriv hældningen for linjestykket

Beregn hældningen for hver af linjestykkerne:

* (3,3) -> (5,3)
* (3,3) -> (5,5)
* (3,3) -> (5,4)

Beregn dernæst hældningen i hånden eller med regnemaskine.

* Er der noget, der undrer dig ved en af resultaterne?
* Overvej, hvorfor det ikke helt er, som du forventer

## Øvelse 3.3: Ny beregning af linjestykkets hældning

I stedet for at bruge heltal (int) til repræsentation af koordinaterne og beregning af hældningen, skal du ændre din implementering, så du kun anvender double-typen til dine data.

Ret koden, og udfør samme beregninger som i forrige øvelse 3.2. Får du samme resultat?

Benyt **Ordet rundt,** og overvej forskellen. Hvad er den mest korrekte implementering af de to?

# Øvelse 4: Manipulation af tekststrenge

Som du har læst og set i dagens forberedelse, så har string-typen forskellige funktioner til at inspicere og manipulere tekststrenge. I de følgende øvelser skal du arbejde med nogle disse funktioner.

Afgør selv, hvilke datatyper er passende til øvelserne.

## Øvelse 4.1: Length()

Afgør længden af en tekststreng:

* Indlæs en vilkårlig tekststreng
* Udskriv tekststrengens længde

Prøv forskellige tekststrenge, og se, om længden passer.

## Øvelse 4.2: Substring()

Udtræk en delstreng fra en givet tekststreng fra en bestemt position:

* Indlæs den fulde streng
* Indlæs positionen
* Udtræk og udskriv delstrengen

Afprøv forskellige tekststrenge og positioner.

Ud over delstrengens position kan du også angive, hvor lang delstrengen skal være.   
Opdatér din kode til:

* Indlæs den fulde streng
* Indlæs positionen
* Indlæs længde af delstreng
* Udtræk og udskriv delstrengen

Afprøv.

## Øvelse 4.3: IndexOf()

Find positionen for en bestemt karakter i en angivet streng:

* Indlæs den fulde streng
* Indlæs karakteren (dvs. bogstavet), der søges efter  
  Vigtigt: Gem karakteren i en variabel af typen char
* Find og udskriv positionen for den angivne karakter, hvis den blev fundet, ellers udskriv beskeden, at karakteren ikke blev fundet

Afprøv.

# Øvelse 5: Brug af Git

Det at kunne dele og versionere din kode er helt afgørende for et godt samarbejde i et team. I skal frem over arbejde med Git, der er et meget udbredt værktøj til netop versionsstyring. Men inden du går i gang, så gennemfør næste terminologi-øvelse.

## Øvelse 5.1: Terminologi

Test 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 ”git init”, ”git pull”, ”git add”, ”git commit”, ”git push”, ”git status” og ”git clone”. 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 fire gruppemedlemmer har bidraget, eller tiden er gået.

*Tidsramme: 20 minutter*

## Øvelse 5.2: Brugen af Git

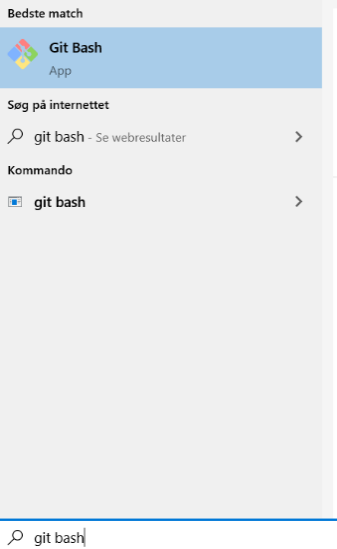
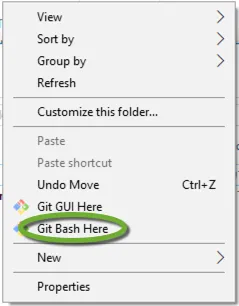
Du forventes på nuværende tidspunkt at have installeret Git og have oprettet en bruger inde på GitLab (med en anden e-mail end jeres studie e-mail, så det kan bruges til fremtidig brug).

**Note:** Det er tilladt at benytte en anden Git repository manager end GitLab, for eksempel GitHub eller BitBucket. Det vigtigste er, at I benytter jer af Git. Men vi beskriver ikke de andre Git repository managers, kun GitLab.

På et senere tidspunkt skal du benytte Git inde fra Visual Studio IDE, men for nu, skal du afprøve Git Bash og få styr på, hvordan man gør alt dette uden brug af et grafisk brugerinterface (en: Graphical user interface).

### Øvelser 5.2.1: Konfigurering af Git

Dette er en individuel øvelse, hvor du skal konfigurere Git på din maskine.

* Benyt **en** af de følgende metoder til at åbne Git Bash:
  + Søg efter ”Git Bash” i din windows menu (nedre billede til venstre)
  + Højreklik i en mappe eller på dit skrivebord og vælg ”Git Bash Here” (nedre billede til højre)
* Tilføj dit brugernavn og din e-mail til den globale git config, benyt følgende kommandoer i Git Bash (husk at erstatte ”*Fornavn Efternavn*” med dit navn, samt ”*DinStudieE-mail@edu.ucl.dk*” med din studie e-mail):
  + git config --global user.name "Fornavn Efternavn"
  + git config --global user.email [DinStudieE-mail@edu.ucl.dk](mailto:DinStudieE-mail@edu.ucl.dk)
* Benyt følgende kommando i Git Bash: git config –list
  + Burde du kunne se følgende i nærheden af bunden af konsollen:

*user.name=Fornavn Efternavn  
user.email=DInStudieE-mail@edu.ucl.dk*

Nu er du klar til næste del af øvelsen.

### Øvelse 5.2.2: Oprettelse af Git repository til hjemmeprojekter

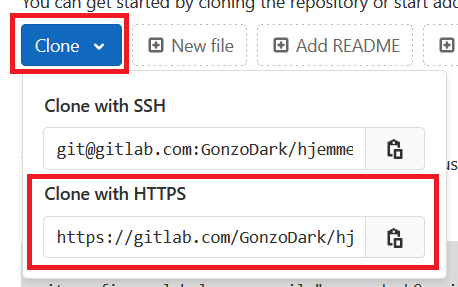
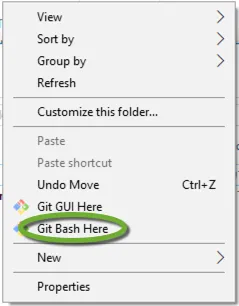
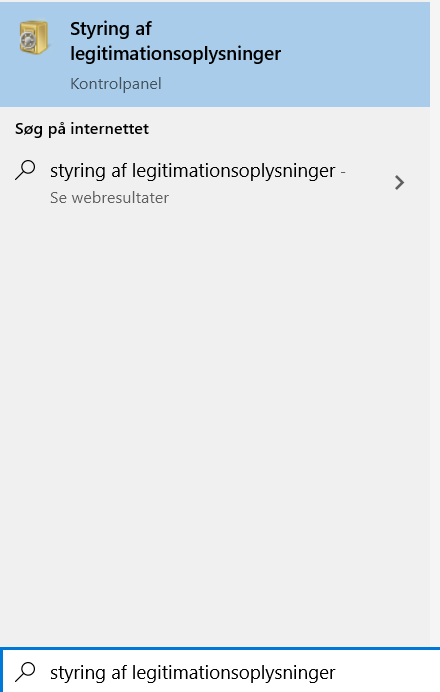
Dette er en individuel øvelse, hvor du skal oprette dit helt eget repository inde på Gitlab til dine hjemmeprojekter.

* Benyt følgende link og opret et nyt projekt inde på GitLab: <https://gitlab.com/projects/new>
  + Vælg ”Create blank project”
    - Giv dit projekt et godt navn for eksempel ”Hjemmeprojekter”
    - Lad dit projekt være ”Private”
    - Undgå at initialisere projektet med en ”README”-fil

Hvis alt er gået som det skulle, bør du nu have en nyt tomt repository, med en beskrivelse af ”Command line instructions”, vi vender tilbage til disse instruktioner længere nede i opgaven.

### Øvelse 5.2.3: Klon dit repository via Git Bash

Du skal nu klone dit netop oprettede Git repository.

* Hvis du ikke allerede er inde og se på dit nyoprettede projekt inde på GitLab, så gå ind på <https://gitlab.com/dashboard/projects> og find dit nye projekt på listen og vælg det
* Tryk på den blå ”Clone”-drop down knap og kopier adressen der står i boksen med ”Clone with HTTPS”, som på billedet nedenunder:
* Find et sted på din computer, hvor du har lyst til at placere dine hjemmeprojekter
* Højreklik i roden af mappen, vælg ”Git Bash Here”, som på billedet nedenunder:
* Benyt følgende kommando: git clone (og den adresse du kopierede tidligere fra GitLab (Clone with HTTPS))
  + For eksempel: git clone https://gitlab.com/DitBrugerNavn/hjemmeprojekter.git
* Nu vil Git kræve at du logger ind, og Windows beder dig om at gemme dine informationer – benyt dit GitLab brugernavn (**ikke** din e-mail) og din GitLab kode.
  + **Hvis** du ved en fejltagelse har indtastet et forkert brugernavn/adgangskode, skal du opdatere dine logininformationer:
    - Åbn Windows-menuen og søg efter ”styring af legitimationsoplysninger” (eller Credential Manager, hvis din udgave af Windows er på engelsk), som på billedet nedenunder:
    - Vælg herefter ”Windows-legitimationsoplysninger”, som på billedet nedenunder:
    - Se under ”Generiske legitimationsbeskrivelser”, lokalisér ”git:https://gitlab.com” og fold punktet ud og, tryk på ”Fjern”, som på billedet nedenunder:
    - Nu er du klar til at indtaste dine legitimationsoplysninger i Git igen

Sådan, nu har du en lokal kopi af dit repository. Denne lokale kopi arbejder vi videre med i næste øvelse.

### Øvelse 5.2.4: Lav en ændring og få den op på GitLab

* Åbn Git Bash inden i din projektmappe
  + Du kan tjekke, om du er inden i mappen ved at benytte følgende kommando: git status
    - Hvis du ikke er inde i en mappe med Git, vil projektet brokke sig, som således:
* Lav en ændring i mappen, benyt følgende kommando: touch README.md
* Nu skal du tjekke, hvordan status er igen, benyt kommandoen: git status
  + Git burde gerne fortælle dig, at du har nogle filer, der ikke bliver fulgt af Git (Untracked files)
* Du skal bede Git om at følge din netop oprettede fil med følgende kommando: git add README.md
* Nu skal du tjekke, hvordan status er igen, benyt kommandoen: git status
  + Nu burde Git sige, at du har ændringer der er klar til at blive committed (Changes to be committed)
    - Dette område er også kendt som ”staging area” og viser dig, hvilke filer der tilføjes, når du opretter dit commit
* Commit filen med: git commit -m “message”
* Nu skal du tjekke, hvordan status er igen, benyt kommandoen: git status
* Nu skal du skubbe filen afsted op til dit remote repository (GitLab), benyt følgende kommando: git push
* Benyt git status igen, samt følgende kommando: git log
  + Bemærk, at du har en enkelt commit med et commit-id, samt en forfatter (author), der indeholder de informationer, som du indtastede i øvelse 5.2.1
* Gå ind på GitLab og bemærk, at du også har dine ændringer på dit remote repository (GitLab)
  + (genopfrisk vinduet)

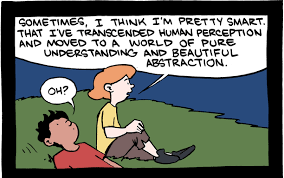
*Tidsramme: 30-45 minutter*

# Loops

|  |  |
| --- | --- |
| **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 |
| **Forventede**  **produkter** | * Implementering af programmeringsøvelserne |
| **Forventet læringsudbytte [SOLO]** | Designsporet:   * [Unistrukturel] Du kan identificere enkelte kendetegn ved **Computational Thinking** – dekomponering, mønstergenkendelse, abstraktion * [Unistrukturel] Du kan genkende enkelte kendetegn ved **strukturmodellering** – klasse, klassenavn, attribut (navn, datatype), operation (navn)   Programmeringssporet:   * [Unistrukturel] Du kan genkende enkelte kendetegn ved C# **datatyper** – int, string, bool, double, char * [Unistrukturel] Du kan huske enkelte kendetegn ved styring af **program flowet** – sekvens, if-else, switch, while, do-while * [Unistrukturel] Du kan genkende enkelte kendetegn i **programstrukturen** – sætning (en: statement), kodeblok (en: code block), løsning (en: solution), projekt i Visual Studio IDE. |
| **Din forberedelse** | Designsporet:   * CT: [Abstraction and Pattern Generalization](https://www.youtube.com/watch?v=RdzYOtxhuDc)  (video: kun de første 6:07) * [Abstraction (BBC)](https://www.bbc.co.uk/bitesize/guides/zttrcdm/revision/1) (3 sider + Test & Score)   Programmeringssporet:   * [YB] 2.3 (s. 33-47) * [Conditionals](https://www.youtube.com/watch?v=-bL3aipO45k) (video: 2:30) * [The C# Switch Statement](https://www.youtube.com/watch?v=BR3E85ilvuA) (video: 9:39) * [Switch Statement in C#](https://www.geeksforgeeks.org/switch-statement-in-c-sharp/) * [C# Switch Examples](http://www.csharp-examples.net/switch/) * [C# While Loop](https://www.youtube.com/watch?v=kf8NK2ol_hE) (video: 7:26) * [C# Do While Loop](https://www.youtube.com/watch?v=N8zEMM3WA68) (video: 6:18) * [Working with Code Files, Projects, and Solutions](https://www.youtube.com/watch?v=ycHtq4DraQo) (video: 11:35) |

Du skal i denne opgave se videre på styring af program flowet med switch-sætningen, endnu en forgrening (en: conditional), og med de to løkketyper (en: loops) while og do-while. Derudover skal du se på abstraktion i CT og arbejde lidt videre med software-designklassebegrebet. **Sørg for at gøre øvelse 2 (konceptuel -> design) i Ex03-CSharpData samt øvelse 3.3 (Menu) i opgaven Ex02-Conditionals færdig som forberedelse, da du arbejder videre med disse øvelser i denne opgave.**

# Dagens ord:



# Øvelse 1: Computational Thinking - Abstraktion

Abstraktion handler i princippet om, at man ud fra en stor mængde information kun fokuserer på de aktiviteter og data, der er relevante for en given opgave. Oftest er det gennem mønstergenkendelse, at man får øje på det relevante elementer, hvorefter man kan abstrahere al ”støjen” væk, dvs. al den unødvendige information.

## Øvelse 1.1: Terminologi

Del teamet i to grupper, og brug CL-strukturen **Møde på midten** til at reflektere over begrebet ”abstraktion”.

1. Reflektér over begrebet individuelt (1-2 min.)
2. Hvert gruppemedlem præsenterer dernæst resultatet af deres refleksion (1-2 min.)
   * Gruppen diskuterer det præsenterede (1-2 min.)
   * Bliv ved, indtil alle gruppemedlemmer er blevet hørt

*Tidsramme: 15 minutter*

## Øvelse 1.2: Identificering af abstraktioner

* Brug 2 minutter individuelt til at identificere abstraktioner, som I allerede har stiftet bekendtskab med i uddannelsens første uger, både i programmering og systemudvikling; der er en hel del.
* Benyt dernæst **Ordet rundt** i teamet i den resterende tid (med taletid til alle), og del jeres fund.

*Tidsramme: 10 minutter*

## Øvelse 1.3: Abstrahér dine klasser

Kig sammen i hele teamet på de klasser, du arbejdede med i den forrige opgave Ex03-CSharpData, både de konceptuelle og software-designklasserne, og se om der er anvendelsessammenhænge, hvor man ville fjerne nogle af attributterne, du har fundet. Bemærk, at det er anvendelsessammenhængene, der oftest styrer hvilke attributter, der er relevante og nødvendige. F.eks. hvis konteksten er et parkeringshus, der registrerer parkerende biler, i modsætning et mekanikerværksted. Har de to kontekster brug for at vide det samme om en bil? Konteksten styrer abstraktionen.

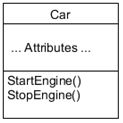
Brug **ordet rundt** til denne øvelse.

*Tidsramme: 10 minutter*

## Øvelse 1.4: Operationer

En klasse beskriver en samling af objekter med præcis de samme egenskaber, selvom egenskaberne typisk har forskellige værdier fra objekt til objekt. Ud over egenskaber (attributter), er det specielt for en software-designklasse også muligt at angive, hvilke operationer designklassen kan have (Bemærk: man angiver aldrig operationer for konceptuelle klasser).

F.eks. for Bil-klassen kan du angive operationer, som objekter af denne klasse kan udføre. Lidt søgt, men helt validt kan en bil starte og slukke sin motor. Dette kan også skitseres i en software-designklasse:



Operationerne StartEngine() og StopEngine() angives med parenteser til sidst og i en sektion nedenunder egenskaberne (attributterne). Både parenteser og egen sektion skal signalere, at der er tale om operationer og ikke egenskaber.

Udfør følgende:

* Gennemgå de tre designklasser Person, Car og Product fra øvelse 2 i den forrige opgave Ex03-CSharpData, og find nye relevante operationer for hver af dem.
* Opdatér dine diagrammer, så operationerne indgår som anvist foroven (med parenteser og ny sektion).

Brug **ordet rundt** til denne øvelse.

*Tidsramme: 15 minutter*

# Øvelse 2: Programmering

De følgende øvelser er en lang række af primært færdighedsopgaver, som dækker stoffet til i dag samt repetition af tidligere emner. Emnerne er blandede, så du kommer godt rundt i stoffet, også selvom du ikke når dem alle. Du kan altid vende tilbage og lave de øvelser, du ikke har nået, når du har tid.

* Lav så mange øvelser, du kan nå, med dagens makker.
* Du skal anvende **parprogrammering**, så husk at skift rolle (driver / navigator) ofte, f.eks. ved hver øvelse. Det er forbudt, at flere programmerer samtidigt.

## Øvelse 2.1: Terminologi

Del teamet op i to mindre grupper, og brug **Ordet rundt** til at reflektere over de tre sætningstyper ”switch”, ”while” og ”do-while”. Sørg for, at alle får mulighed for at tale.

*Tidsramme: 10 minutter*

## Øvelse 2.2: Forgrening med Switch

Switch er den anden forgreningssætning (en: conditional statement) ud af to, hvor den første er if-else. Switch og if-else kan det samme, men afhængigt af hvad man vil opnå (øget læsbarhed, færre linjer, mm.), har de to syntakser fordele og ulemper. Det lærer du efterhånden ved at bruge dem.

* Tilpas din kode fra øvelse 3.3 (Menu) i den tidligere opgave Ex02-Conditionals med brug af switch-sætningen i stedet for if-else-sætningen. Programmet udskriver en simpel menu samt en besked afhængig af menuvalg.

## Øvelse 2.3: Debugging igen

Det kan ikke siges for ofte: Har man en fejl i sin kode, som er svær at finde, kan man med fordel debugge (”afluse”) koden i stedet for at køre hele koden på en gang. At debugge betyder, at man kører programmet trin for trin, dvs. sætning for sætning. Man kan derved følge programmets flow nøjere, overvåge hvordan hver variabel skifter værdi, og se, hvor fejlen opstår i koden. **Debugging er en meget stærk teknik, som du anbefales at anvende hele tiden (essentiel færdighed), så du hurtigst muligt får det på rygraden.** Udover at finde fejl, så kan debugging altid hjælpe en med at forstå kode hurtigere og måske få ideer til at optimere, specielt hvis en anden har skrevet koden.

I denne øvelse skal du prøve at gå trinvist igennem jeres kode med funktionstasten **F10 (Step Over)**. Hvis denne funktionstast ikke er konfigureret til at fungere på denne måde, så sig til.

Udfør følgende:

* Debug din kode fra forrige øvelse 2.2 ved at starte med at trykke F10 (Step Over) for at starte debugging. Programmet går i gang og stopper på der første mulige trin i koden; i denne opgave er det starten af Main()-metoden.
* Klik nu på Locals-fanen for at kunne løbende følge med i ændringer af variable mm.
* Med F10 gå igennem koden trin for trin.

Du kan på ethvert tidspunkt inspicere alle de variable, som er tilgængelige, hvor koden er stoppet, bl.a. også ved at sætte musepilen over et variabelnavn i koden.

## Øvelse 2.4: Læs enkelte karakterer i en tekststreng

En tekststreng (repræsenteret ved datatypen **string**) er i realiteten en ordnet samling af karakterer, hvor hver karakter er af typen **char**. Man kan tilgå hvert enkelt element i en string-samling ved at benytte notationen:

* <variabelnavn> ”[” + <index> + ”]”, hvor <index> er et heltal fra 0 og opefter.

Index angiver den position i tekststrengen, man ønsker at læse. Bemærk at første element har index 0.

Eksempelvis hvis man ønsker at læse den fjerde karakter ’d’ i nedenstående string-variabel str:

string str = "abcdefg";

skriver man:

char ch = str[3]; // 'd'

Bemærk, man kan kun læse en karakter fra en tekststreng med [ ]-notationen og aldrig sætte den. Det gøres på andre måder.

Implementér følgende:

* Indlæs en vilkårlig tekststreng (på mindst seks karakterer)
* Læs og udskriv specifikt det andet, fjerde og sjette element (dvs. karakter) i tekststrengen

Debug koden med F10 (Step Over).   
Når alle fejl er fundet og rettet, kør koden med F5.

## Øvelse 2.5: While-løkke: Gennemløb alle karakterer i en tekststreng

Udskriv alle karakterer i en tekststreng kun ved hjælp af en while-løkke:

* Indlæs en vilkårlig tekststreng
* For hver enkel karakter i tekststrengen:
  + Udskriv karakteren på formen ”<index>: ’<karakter>’”

Eksempel:

* Tekststrengen ”hej” bliver til de 3 linjer:  
  0: ’h’  
  1: ’e’  
  2: ’j’

Gode overvejelser til dig, før du begynder at kode:

* Tekststrengen kan jo have vilkårlig længde, da du ikke ved, hvad brugeren vil indtaste. Det skal du tage højde for i koden. Det kan bl.a. være godt at finde ud, hvor lang den indtastede tekststreng er, inden man udskriver hver enkel karakter. While-løkken styres jo af en betingelse, der afgør om løkkens kodeblok skal udføres eller ej. Overvej først algoritmisk, hvordan du styrer det rigtige antal aktiveringer af kodeblokken, før du begynder at kode.

Debug koden med F10 (Step Over).   
Når alle fejl er fundet og rettet, kør koden med F5.

## Øvelse 2.6: Do-while-løkke: Gennemløb alle karakterer i en tekststreng

Implementér nu forrige øvelse 2.5, men denne gang udelukkende ved hjælp af en do-while-løkke.

Overvej forskellen, og om der kan opstå problemer for visse tekststrenge.

## Øvelse 2.7: Heltalsdivision

Du har allerede tidligere (i opgave Ex03-CSharpData, øvelse 3.3) oplevet, at division med to heltal ikke helt går, som du måske først forventede det. Forhåbentlig har du opdaget, at:

* Dividerer man to heltal (int-datatypen) med hinanden i C#, bliver resultatet uundgåeligt også et heltal, dvs. datatypen int.

Derfor kan 3/2 ikke blive til 1,5 men kun til heltalskvotienten 1, da resultatet skal være et heltal.

Så 2 går op én gang i 3, men hvordan finder man så rest-delen?

* Åben google, og søg efter: ”C# remainder operator”.

Benyt den fundne operator til at implementere følgende:

* Indlæs første heltal
* Konvertér og gem heltalsværdien i int-variablen a
* Indlæs andet heltal
* Konvertér og gem heltalsværdien i int-variablen b
* Beregn og gem (i passende variabel) heltalskvotienten ved heltalsdivisionen a/b
* Beregn og gem (i passende variabel) rest-delen ved heltalsdivisionen a/b
* Udskriv den fundne heltalskvotient og rest-del

Debug koden med F10 (Step Over).   
Når alle fejl er fundet og rettet, kør koden med F5.

## Øvelse 2.8: Hold programmet kørende, indtil brugeren afslutter det

Stort alle øvelser indtil nu har fulgt mønstret:

1. Indlæs data
2. Beregn eller behandl data
3. Udskriv resultat

Efter disse tre trin afsluttes programmet, hvilket betyder, at du skal starte programmet igen for at prøve med nogle nye data. Det er ret irriterende, hvis du gerne vil prøve med flere data, så det må du gøre noget ved.

Du skal fremover anvende følgende korrigerede mønster:

1. Så længe brugeren ønsker at indtaste data, da:
   1. Indlæs data
   2. Beregn eller behandl data
   3. Udskriv resultat

Overvej, hvilken type af program flow der kan realisere dette mønster?

Udfør følgende:

* Opdatér koden i øvelse 2.2 (med menuen), så det bruger det nye mønster, og man bliver i programmet, indtil en passende slutbetingelse er opfyldt. Find selv på en slutbetingelse, som brugeren kan vælge til at afslutte programmet, dvs. udtrykker at brugeren IKKE ønsker at indtaste mere data, men afslutte.
* Det kan i øvrigt være fornuftigt at ’rense’ konsolvinduet for indhold, før brugeren indlæser nye data. Indtast ”Console.” (husk punktum) i editoren og brug IntelliSense til at finde en passende funktion til netop dette.

Brug det nye mønster i de følgende øvelser, men vær opmærksom på, at en slutbetingelse nok skal angives på forskellige måder afhængig af programmets funktion.

## Øvelse 2.9: Udskriv hver anden karakter i en tekststreng

Tag udgangspunkt i din kode fra øvelse 2.5, og udskriv denne gang udelukkende hver anden karakter i tekststrengen. Hint: remainder-operatoren kan måske bruges J

Debug koden med F10 (Step Over).   
Når alle fejl er fundet og rettet, kør koden med F5.

## Øvelse 2.10: Tekststrenge med cifre

Udskriv alle forekomster af cifre, dvs. karaktererne ’0’ til ’9’, du finder i en tekststreng.

* Indlæs en vilkårlig tekststreng (indsæt cifre hist og pist)
* For hver enkel karakter, som er et ciffer i tekststrengen:
  + Udskriv ciffer-karakteren på formen ”<index>: ’<karakter>’”

Debug koden med F10 (Step Over).   
Når alle fejl er fundet og rettet, kør koden med F5.

## Øvelse 2.11: Tekststrenge med cifre og operatorer

Udskriv alle forekomster af cifre, dvs. karaktererne ’0’ til ’9’ samt operatorerne ’+’ og ’-’, du finder i en tekststreng.

* Indlæs en vilkårlig tekststreng (indsæt cifre hist og pist)
* For hver enkel karakter i tekststrengen:
  + Hvis karakter er et ciffer:
    - Udskriv ciffer-karakter på formen ”<index>: ’<karakter>’ (ciffer)”
  + Hvis karakter er en operator ’+’ eller ’-’
    - Udskriv operator-karakter formen ”<index>: ’<karakter>’ (operator)”
  + Ellers udskriv karakteren på formen ”<index>: ’<karakter>’ (ukendt)”

Overvej, hvilken forgreningssætning (en: conditional) egner sig bedst her: Evt. prøv begge.

Debug koden med F10 (Step Over).   
Når alle fejl er fundet og rettet, kør koden med F5.

## Øvelse 2.12: Skan og udregn simple regneudtryk (lidt sværere)

Med udgangspunkt i koden fra forrige øvelse 2.11, skal brugeren nu kunne angive et regneudtryk med cifrene ’0’ til ’9’ samt operatorerne ’+’ og ’-’. Koden skal skanne tekststrengen med regneudtrykket, finde cifre og operatorer og beregne værdien af regneudtrykket, der angives. F.eks.:

* Tekststrengen ”4-5+9” skannes og udregnes til 8
* Tekststrengen ”7+4-3+6” skannes og udregnes til 14

Dette sætter selvfølgelig nogle krav til et gyldigt regneudtryk:

* Et udtryk skal starte og slutte med et ciffer
* Et ciffer skal efterfølges af en operator med mindre, at det er det sidste ciffer i udtrykket
* En operator skal følges af et ciffer
* Alle karakterer, der ikke er et ciffer, ’+’ eller ’-’, er ugyldige.   
  Sørg for at meddele brugeren, at der er angivet et ugyldigt udtryk, hvis en ugyldig karakter forekommer i udtrykket.

Disse krav sætter de fleste emner, vi har set på, i spil, både i C# og i Computational Thinking.

Udbyg koden fra øvelse 2.11.  
Debug koden med F10 (Step Over).  
Når alle fejl er fundet og rettet, kør koden med F5.

## Øvelse 2.13: Multiplikation og division (endnu sværere)

Hvis du bliver færdig med øvelse 2.12, inden tiden er gået, da implementer også operatorerne ’\*’ (multiplikation) og ’/’ (division) i regneudtrykket.