# Methods (da: Metode)

|  |  |
| --- | --- |
| **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** | * En algoritme der beskriver, hvordan man laver en kransekage via Computational Thinking tilgangen (systemisk tænkning) * Et C# konsolprogram der indeholder basale lommeregner funktioner (plus, minus, gange og divider) |
| **Forventet læringsudbytte [SOLO]** | Designsporet:   * [Unistrukturel] Du kan identificere enkelte kendetegn ved **Computational Thinking** – dekomponering, mønstergenkendelse, abstraktion, algoritme * [Unistrukturel] Du kan genkende enkelte kendetegn ved **strukturmodellering** – klasse (navn), attribut (navn, datatype, synlighed (en: visibility)), operation (navn, parameter, returtype, synlighed), association, Design Class Diagram (DCD)   Programmeringssporet:   * [Unistrukturel] Du kan genkende enkelte kendetegn ved **OOP-principper** – indkapsling (en: encapsulation) * [Unistrukturel] Du kan genkende enkelte kendetegn ved **C#** **klasse** – C# klasse, felt (en: field), metode, access modifiers (public, private) * [Unistrukturel] Du kan genkende enkelte kendetegn i den **logiske** **programstruktur** – sætning, kodeblok, metode (navngivet sekvens), klasse * [Unistrukturel] Du kan huske enkelte kendetegn ved **programflow** – sekvens, forgrening (if-else, switch), løkke (while, do-while, for), metodekald og -retur   Færdighedssporet:   * [Unistrukturel] Du kan huske enkelte kendetegn ved **versionering** – git, git-kommandoer. * [Unistrukturel] Du kan huske enkelte kendetegn ved **debugging** |

|  |  |
| --- | --- |
| **Din forberedelse** | Designsporet:   * CT:   + [Computational Thinking: Algorithm Design](https://www.youtube.com/watch?v=N91oCQbWUvA) (video: 14 min)   + [BBC Algorithms](https://www.bbc.co.uk/bitesize/guides/zpp49j6/revision/1)   + Kom i gang med algoritmisk design.mp4 (video: 11:31; se i planen i ItsLearning)   Programmeringssporet:   * Læs om while, do-while og for-loop i [YB], side 39 og 42 (til og med ’For loop’) * Programming Foundations: Object-Oriented Design:   + Bemærk, I skulle meget gerne alle sammen være oprettet inde på LinkedIn Learning med jeres studie login (hvis ikke, da kontakt IT-service)   + 1. Object-Oriented Fundamentals:     - [Object-oriented thinking](https://www.linkedin.com/learning/programming-foundations-object-oriented-design-3/object-oriented-thinking?autoplay=true&resume=false&u=57075649) [video: 4:23]     - [Objects](https://www.linkedin.com/learning/programming-foundations-object-oriented-design-3/objects?u=57075649) [video: 5:01]     - [Classes](https://www.linkedin.com/learning/programming-foundations-object-oriented-design-3/classes?u=57075649) [video: 4:38]     - [Encapsulation](https://www.linkedin.com/learning/programming-foundations-object-oriented-design-3/encapsulation?autoplay=true&resume=false&u=57075649) [video: 3:38]   + 5. Class Diagrams:     - [Creating class diagrams: Attributes](https://www.linkedin.com/learning/programming-foundations-object-oriented-design-3/creating-class-diagrams-attributes?u=57075649) [video: 2:24]     - [Creating class diagrams: Behaviors](https://www.linkedin.com/learning/programming-foundations-object-oriented-design-3/creating-class-diagrams-behaviors?u=57075649) [video: 3:09]     - [Converting class diagrams into code](https://www.linkedin.com/learning/programming-foundations-object-oriented-design-3/converting-class-diagrams-into-code?u=57075649) [video: 4:29]     - [Instantiating classes](https://www.linkedin.com/learning/programming-foundations-object-oriented-design-3/instantiating-classes?u=57075649) [video: 3:39] * Læs om ’methods’ i [YB] til og med 3.1.6, side 48 til 54 * [Access Modifiers (C# Programming Guide)](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/access-modifiers)   + Med fokus på public og private * Læs om ’void’ i [YB], side 15 * Læs ’Getting Started’ i [PG], side 9 til 17 (til og med ’The Three States’) * Genlæs [parprogrammering](https://da.wikipedia.org/wiki/Parprogrammering)   Færdighedssporet:   * Skimlæs [navigating through code with the debugger](https://docs.microsoft.com/en-us/visualstudio/debugger/navigating-through-code-with-the-debugger?view=vs-2019) |

# Beskrivelse af dagens opgave

I dagens opgave skal du arbejde videre med Computational Thinking, specielt processen ved at etablere en algoritme. Opgaven tager udgangspunkt i en madopskrift, som har et vist niveau af struktur, men ikke en formaliseret beskrivelse med algoritmens grundelementer, såsom: trin, valg, gentagelse og betingelser (for valg og gentagelser).

Derudover vil du blive introduceret til C# klasser, felter og metoder.

**Sørg for at lave øvelse 3 (Lommeregner) i denne opgave færdig, da du skal arbejde videre med denne øvelse i næste opgave Ex06-UnitTestAndArray.**

# Dagens ord: Konceptuel klassediagram versus Software klassediagram

Henover de sidste par opgaver har vi arbejdet med to forskellige klassediagrammer. Det ene klassediagram har fokus på at beskrive det konceptuelle niveau, også kaldet en domænemodel i [Larman][[1]](#footnote-1). Det andet er i programmeringsopgaverne beskrevet som en samling af ’software-designklasser’ med tilhørende attributter, operationer og associationer; i [Larman] betegnet som et designklassediagram (en: Design Class Diagram eller helt kort DCD). Begge diagrammer (domænemodel og DCD) skitseres/tegnes ved brug af UML-notationen, hvor UML står for Unified Modeling Language (en ISO standard), og begge udtrykkes via UML-diagramtypen statisk klassediagram. ’Statisk’ henviser til, at der beskrives information/data og ikke adfærd.

Når man implementerer en DCD i C#, er der oftest en tydelig (hvis ikke 1-til-1) navngivningsmæssig sammenhæng mellem DCD’ens designklasser og de tilsvarende C#-klasser i din kode. Attributter i en DCD-klasse omdannes til felter (en: fields) (også kendt som instansvariable) i en C#-klasse, og operationer (klassens funktionalitet) omdannes til metoder i en C#-klasse[[2]](#footnote-2).

# Øvelse 1: Blomsterberg-kransekage til nytårsaften

Du skal forestille dig, at det er nytårsaften, og du vil lave følgende kransekage:

Opskriften, som du skal benytte i de følgende øvelser, kan du finde her med fremgangsmåde og ingredienser:

1. <https://www.dr.dk/mad/opskrift/kransekagetop-flot-kransekage-til-nytarsaften>

## Øvelse 1.1: Analysér opskriften

Brug **Møde-på-midten** til at analysere trin og tekst i opskriftens fremgangsmåde. Overvej, hvad du skal være opmærksom på i teksten med Computational Thinking i baghovedet (f.eks. mulige uklarheder, tvetydigheder, alternativer, valg, gentagelser, underforståede trin, mm.).

Fremgangsmåde:

* Individuelt: 5 minutter
* fælles (Ordet Rundt én gang): 5 minutter

## Øvelse 1.2: Opbyg algoritme i gruppen

Brug **Tænk-par-del**, og omdan opskriften til en mere formel algoritme.

Brug beskrivelsesformen for en use case til at beskrive algoritmen (en nummeret punktform). Overvej, hvordan du viser trin, gentagelser, valg og betingelser. Kig i Larman på s. 69, med fokus på ”Main Success Scenario” (det er en del af afsnit 6.8: ” Use Case UC1: Process Sale”) for et praktisk eksempel (**use cases er introduceret i systemudvikling i denne uge**).

Start med første punkt i kransekagemassen og nå så langt, du kan på den tid, der er; dog ***få minimum de to første trin med*** i opskriften på kransekagemassen med i algoritmen:

1. Individuelt: brug 10 minutter til at skrive de trin ned, som du vurderer, at der kan indgå i en algoritme
2. Parvis: brug 10 minutter til at diskutere hver jeres algoritme
3. Gruppen: brug 10 minutter (eller mere) og diskutér jeres algoritmer (Ordet Rundt), og bliv enige om en fælles algoritme.

Skriv jeres beslutninger ned.

## Øvelse 1.3: Del jeres algoritme, søg inspiration, giv feedback

Gennemfør en **Tre-til-te** CL-struktur, hvor dit team sender medlemmer afsted til de andre teams, et medlem per team, og én bliver tilbage og præsenterer jeres begrundede algoritme (use case, main success scenario) til de besøgende.

Herefter skal du og dit team, når dem, der har været på besøg, er kommet tilbage, sammenligne jeres algoritme og begrundelser med det, de andre besøgte teams har produceret. Hvis der er behov for det, giv eller efterspørg feedback, tilpas jeres algoritme (*vend tilbage til dette punkt, hvis I er de første, der er færdige*).

*Tidsramme: max 15. minutter*

# Øvelse 2: Terminologi

Inden du skal i gang med dagens konkrete programmeringsopgaver, skal du have testet din forforståelse af dagens emne (samt fastholdt et par af de tidligere nævnte emner).

Del teamet op i 2 grupper, og brug CL-strukturen **tænk-par-del** (se nedenstående fremgangsmåde) til at reflektere over begreberne:

1. *C# metode*
   * *Med fokus på: access modifier, returtype, metodenavn, parameter*
2. *C# klasse, attributter, objekt (en: object), felt, konstruktør (en: constructor)*
3. *Object-Oriented Programming (OOP)*
   * *Med fokus på: indkapsling (en: encapsulation)*
4. *while, do-while, for-loop*

**Fremgangsmåde:**

1. Team: Del punkterne ud mellem de to grupper (1 minut)
2. Individuelt: Skriv dine overvejelser ned om begreberne (5 minutter)
3. Par: Del jeres tanker med hinanden i gruppen (2 minutter)
4. Par: Forbered jer på, hvad der skal deles (2 minutter)
5. Team: Præsentere på skift det, som I er blevet enige om, til hele teamet (uret rundt) (2 minutter per gruppe)
   1. De øvrige omkring bordet tager noter og stiller spørgsmål

*Tidsramme: 15 minutter*

# Øvelse 3: Lommeregner

Du skal i denne øvelse inspicere en simpel use case og tilhørende designklassediagram (DCD). Herefter skal du oprette et lommeregnerprojekt med en lommeregner-klasse. **Undervejs skal du benytte dig af debugging**.

## Øvelse 3.1: Inspicér use case og designklassediagram (DCD)

Du og dit team skal benytte jer af en passende CL-struktur til at inspicere følgende use case og designklassediagram.

### Use case: Udregn regnestykke

**Primæraktør:** Beregneren

**Main Success Scenario:**

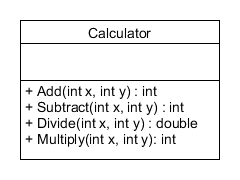
1. Beregneren modtager et regnestykke
2. Beregneren starter ny beregning
3. Beregneren indtaster tal
4. Beregneren vælger regneoperator (+, -, \*, /)

*Beregneren gentager trin 3 og 4, indtil hele regnestykket er indtastet*

1. Systemet beregner og viser resultatet

### Designklassediagram: Calculator

Med fokus på en enkelt software designklasse[[3]](#footnote-3):



## Øvelse 3.2: Lommeregnerprojekt

**Benyt parprogrammering**.

I denne øvelse skal du oprette og benytte en simpel lommeregner. Hertil skal du oprette et lommeregnerprojekt, din første C# klasse, dine første C# metoder, samt benytte dig af en ny debugging kommando (Step Into).

### Øvelse 3.2.1: Opret lommeregnerprojekt og lommeregner klasse

1. Opret en ny konsolapplikation
   1. Navngiv det ”Lommeregner”
2. Tilføj en ny C# klasse til din konsolapplikation
   1. Højre-klik på dit projekt ”Lommeregner” i din Solution Explorer, vælg Add, vælg Class, giv klassen navnet ”Calculator”.

### Øvelse 3.2.2: Opret lommeregner metoder

1. Du skal nu i Calculator-klassen oprette metoderne ”Add”, ”Substract”, ”Divide” og ”Multiply” (se også ovenstående DCD fra øvelse 3.1)
   * Add, skal lægge to tal sammen og returnere dette som et heltal
   * Subtract, skal trække to tal fra hinanden og returnere dette som et heltal
   * Divide, skal dividere to tal og returnere dette som et decimaltal
   * Multiply, skal gange to tal med hinanden og returnere dette som et heltal

### Øvelse 3.2.3: Benyt lommeregner

1. Du skal nu skifte over til din Main-metode i din Program-klasse (Program.cs). Her skal du lave en ny instans af din Calculator-klasse og benytte en af objektets metoder og printe resultatet ud på konsolvinduet

### Øvelse 3.2.4: Benyt debugging

Du har allerede fået introduceret dette vigtigt redskab i tidligere, men denne gang skal du benytte dig af ’Step Into’, frem for ’Step Over’.

1. Benyt [debugging](https://docs.microsoft.com/en-us/visualstudio/debugger/navigating-through-code-with-the-debugger?view=vs-2019) ’Step Into’ (F11)

# Øvelse 4: Mere menu

Du skal overveje ud fra den resterende tid, om du og dit team har tid til denne øvelse.

## Øvelse 4.1: Opret menu med hjælp af en do-while løkke

1. Hvis du stadig har tid, så lav en menu i konsolapplikationen, benyt do-while løkke.
   * Det skal være muligt at benytte alle metoderne på skift.

## Øvelse 4.2: Indtaste tal og modtage svar

Du skal gøre det muligt for en bruger at vælge en af metoderne (Add, Multiply, Divide eller Substract), samt indtaste to tal og modtage svar tilbage på skærmen.

**Krav til implementeringen:**

* Du skal benytte følgende datatyper:
  + double
  + int
  + String
* Benyt Console.ReadLine();
* Benyt ‘if-else’ eller en ‘switch’
* Du skal sikre dig imod forkert input

# Øvelse 5: Review-spørgsmål

* Skal man definere en datatype ved metodedefinitionen (formelle)?
* Kan man have mere end en inputparameter?
* Hvad tænker man på ved returtypen?
  + Hvordan sikrer man at metoden returnerer noget?
  + Hvad hvis man ikke vil returnere noget?
* Skal man definere en datatype ved metodeskaldet (aktuelle)?
* Kan man kalde ens metode mere end en gang?
* Er der forskel på ”git add” og ”git commit”?
* Hvad er forskellen mellem ”git pull” og ”git push”?
* Hvad gør ’Step Into’?
* Hvad gør ’Step Over’?

1. Se “16.2 Definition: Design Class Diagram” i [Larman], side 251 [↑](#footnote-ref-1)
2. Se “16.6 Operations and Methods” I [Larman], side 257 [↑](#footnote-ref-2)
3. Se “The Meaning of *Class* in Different Perspectives” i [Larman], side 13 [↑](#footnote-ref-3)