

## Øvelse 1:

Det er tanken med disse underøvelser, at du bliver lidt klogere på forskellene mellem at arbejde med og uden arrays, samt snuse lidt til exception handling og parsing. Tidligere introducerede emner (fra tidligere programmeringsopgaver) vil også komme i spil.

Du skal i underøvelserne gå sammen med din sidemand og benytte parprogrammering (Ja, det vil sige, at én er driver og én er navigatør/observer!).

### Øvelse 1.1: Alderen på alle i gruppen, samt aldersgennemsnit (uden array)

Din opgave er at printe alderen af alle ved bordet samt printe gennemsnitsalderen ud til skærmen. Du må **ikke** benytte dig af et array til denne opgave.

#### Fremgangsmåde:

- Erklær en række af variabler til at indeholde alderen for hver person ved dit bord
- Tildel variablerne (aldrene) retvisende værdier
- Print aldrene ud til skærmen, en alder per linje
- Erklær en variabel til at kunne indeholde gennemsnitsalderen af alle ved dit bord
- Beregn og print gennemsnitsalderen ud til skærmen

### Øvelse 1.2: Alderen på alle i gruppen, samt aldersgennemsnit (med array)

Denne øvelse foregår på samme måde som sidste øvelse (3.1). Du skal modsat sidste opgave benytte dig af et array til denne opgave samt en passende løkke (en: loop).

Bemærk, at du i det følgende **ikke** må anvende de indbyggede funktioner Sum() og Average() i array-typen til at løse opgaven. Dvs. du skal selv implementere beregningen af aldersgennemsnittet.

#### Fremgangsmåde:

Erklær et array med en passende størrelse, så det kan indeholde alle aldre ved bordet (og af en passende datatype)

Tilføj alderen af alle ved bordet til arrayet

- Erklær en variabel til at kunne indeholde gennemsnitsalderen af alle ved dit bord
- Benyt en løkke (en: loop) til at printe alle aldrene fra arrayet ud til skærmen
- Beregn og print gennemsnitsalderen ud til skærmen

Overvej herefter: Er der en forskel i længden af koden i denne øvelse (1.2) kontra den tidligere (1.1)?

#### Øvelse 1.2.1: Erklær og tildel

Inspicér følgende artikel med henblik på det første kodeeksempel: [Arrays](#)

Lav herefter om på dit program, så du både erklærer og tildeler dit arrays elementer i samme omgang.

### Øvelse 1.3: Findes der en med alderen

Bemærk, at du i det følgende **IKKE** må anvende den indbyggede funktion Contains() i array-typen til at løse opgaven. Dvs. du skal selv implementere søgningen.

Din opgave er at udvide programmet fra ovenstående øvelse (1.2.1). Du skal tilføje en søgefunktion, hvor det er muligt at søge efter, om der findes en bestemt alder i arrayet. Under din implementering skal du benytte dig af "break".

Når du er færdig med implementeringen, skal du overveje, hvorfor det er smart at benytte et break – diskuter dette med din sidemand.

### Øvelse 1.4: Bestemt antal personer

Udvid løsningen fra øvelse 1.3. Det skal være muligt for en bruger af dit program at definere antallet af personer, der er i en givet gruppe – for dernæst at tilføje deres aldre. Du skal med andre ord erklære en variabel til at indeholde brugerinput omkring antallet af personer i en gruppe, for derefter at sætte størrelsen af dit array til at være lig med denne variabel.

#### Øvelse 1.4.1: try-catch

Du skal benytte try-catch til at håndtere, hvis en bruger skriver bogstaver, frem for heltal.

#### Øvelse 1.4.2: int.TryParse

Frem for at benytte dig af try-catch, som i øvelse 1.4.1, skal du denne gang læse omkring [int.TryParse](#) og benytte det i din løsning.

## Øvelse 2: Terminologi

Inden du skal i gang med opgavens programmeringsøvelser, skal du have testet din forforståelse af nogle af emnerne, du har set på tidligere.

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

1. *Designklasse: navn, attribut, operation, synlighed (en: visibility)*
2. *C# klasse: felt, metode, constructor, instantiering*
3. *Program flow: sekvens, forgrening, løkke, metodekald/-retur*
4. *Datahåndtering: variable, tildeling, udtryk*
5. *Datatyper: int, double, string, char, bool, array*

#### Fremgangsmåde:

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

*Tidsramme: 20 minutter*

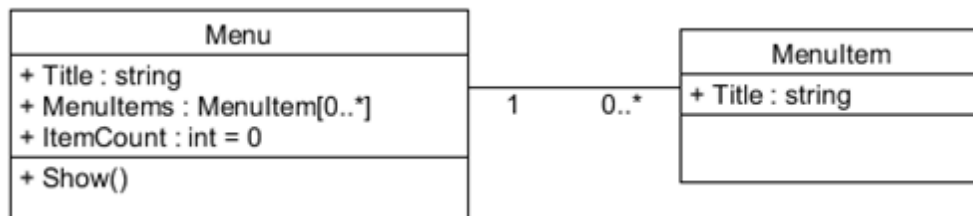
## Øvelse 3: Menusystem – DCD til C# kode

Benyt **parprogrammering** til alle øvelser. Husk at skifte, så I begge får mulighed for at kode.

Du skal nu sammen med din makker tage udgangspunkt i en færdig DCD, som beskriver et simpelt menusystem, og udvikle den tilsvarende kode i en konsolapplikation i Visual Studio. For hver øvelse i det følgende udvikles menusystemet trinvist startende med et simpelt design, som du derefter bygger mere og mere på. DCD'en samt tilhørende beskrivelser skal opfattes som en designspecifikation, som du nøje skal overholde (for at blive betalt af DesignAway); så lav det i hvert trin, du bliver bedt om af DesignAway, og ikke det, som du tror, der skal laves ☺

### Øvelse 3.1: Første implementering – Opret og vis menu

Du har fået nedenstående DCD udleveret fra softwaredesignfirmaet DesignAway, som du nu skal implementere i C#. Kig nøje på diagrammet og dens klasser, attributter og operationer. Husk at overholde den navngivning, der er anvendt forneden i din C#-kode:



*Bemærk at + (plus) eller evt. - (minus) tegnet foran hver attribut og operation angiver elementets synlighed (en: visibility), hvor + betyder synlig, og - (minus) betyder privat eller usynlig uden for klassen.*

I DCD'en er alt angivet som synligt (public). Det skal du også sørge for, når du implementerer i C#.

Menu-klassen har følgende indhold:

- **Title:** (public) indeholder en overordnet titel på menuen. Typen er string.
- **MenuItems:** (public) indeholder alle menuens menupunkter.

*Bemærk UML-notationen i typen "MenuItem[0..\*]" angiver multipliciteten (antallet) af typen MenuItem inden i de kantede parenteser. **Læs det sådan:** attributten "MenuItems" indeholder nul eller flere instanser (objekter) af typen MenuItem. Dette er også angivet i associationen (stregen) mellem de to klasser, dvs. Menu-klassen indeholder en samling af MenuItem-objekter med det angivne antal. \* betyder her flere, men ikke uendelig.*

Husk at DCD'en er en designmodel og ikke behøver at bestemme, hvorledes denne samling af MenuItem-objekter faktisk implementeres i C#, dvs. programmøren har frihed til at vælge den bedst egnede datastruktur til dette. **I din C# kode vælger du dog denne gang at anvende et array til at implementere denne 1-til-mange association**, da du endnu ikke har lært andre måder. Husk i C# bliver du nødt til at definere den maksimale størrelse på array'et.

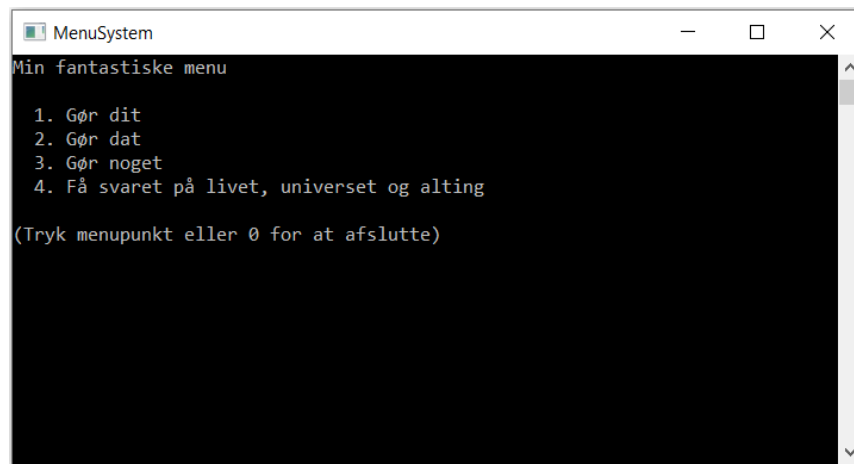
- **ItemCount:** (public) indeholder antallet af menupunkter, dvs. antallet af MenuItem-objekter oprettet i MenuItems-attributten. Typen er int. Denne værdi har du brug for til at holde styr på,

når du indsætter nye menupunkter i din menu.

Bemærk at ItemCount initialiseres til værdien 0 til at starte med (se i DCD'en).

Udfør nu følgende:

- Opret en ny konsolapplikation med projektskabelonen 'Console App', og giv projektet navnet "MenuSystem".
- Når du om lidt kører programmet (du er ikke færdig endnu), vil du gerne have, at konsolvinduet kommer (nogenlunde) til at se ud som følger:



```
MenuSystem
Min fantastiske menu

1. Gør dit
2. Gør dat
3. Gør noget
4. Få svaret på livet, universet og alting

(Tryk menupunkt eller 0 for at afslutte)
```

Menu-titlen står øverst. Menupunkter listes neden under, og en hjælpetekst vises til sidst, så brugeren ved, hvad der skal gøres. Vink: brug `Console.Clear()`-metoden til at "rense" skærmen.

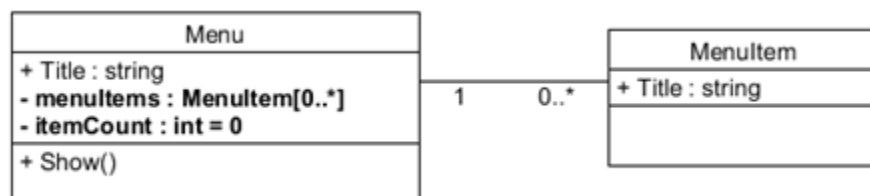
- Implementér Show()-metoden, så den viser ovenstående (selvfølgelig ud fra værdierne i klassens egne felter)
- Kør programmet, og tjek om konsolvinduet passer med ovenstående.
- Hvis ikke, så ret, og prøv igen.

### Øvelse 3.2: Indkapsling

Når du kigger på programmet, så er der en ting, som springer i øjnene:

- Lige nu kan brugerne af Menu-klassen, dvs. i dette tilfælde Main()-metoden, ændre alle felter i klassen, da de alle er public. Dette inkluderer itemCount, der ved et uheld kan sættes til en værdi, der ikke passer med de faktiske antal menupunkter i MenuItems. Dette er et designmæssigt problem, som kunne føre til, at programmet stopper u hensigtsmæssigt.

Du vil gerne bedre kunne kontrollere dette og ønsker ikke, at brugerne af menusystemet skal kunne se, hvordan du har implementeret din interne datastruktur i Menu-klassen, ej heller ændre noget i den. Du skal derfor indkapsle visse dele, så MenuItem-objekter og itemCount ikke længere er tilgængelige uden for Menu-klassen. DCD'en ændres nu til følgende (vist med fed skrift):



Her er synligheden af menuItems og itemCount ændret til privat ved at angive et - (minus) tegn. Bemærk også, at attributternes startbogstav er ændret fra stort til lille. Dette kommer vi ind på senere, men har noget med navngivningskonventioner at gøre.

*Husk, at indkapsling er et af de fire OOP-principper. Du ser her med denne ændring et eksempel på, hvordan dette princip kan implementeres konkret i C# kode.*

Udfør følgende:

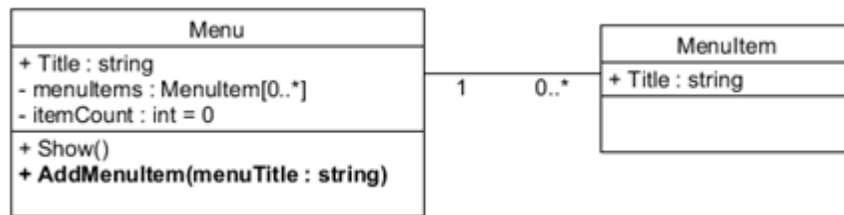
- Implementér denne ændring i din Menu-klasse i C# med brug af access modifiers. Husk også at ændre til små startbogstaver.
- Kompilér programmet, og se så, hvad der sker.
- Overvej med din makker, hvad problemet er.

Indkapslingen betyder, at du ikke længere har adgang til Menu-klassens felter, hvilket jo også var meningen, da de skulle beskyttes. Men det er ikke nok bare at gøre din datastruktur privat, da du jo stadig skal have mulighed for at sætte dem.

- Benyt **ordet rundt**, og overvej kort, hvordan kan du så kan tilgå data i Menu-klassen på en mere kontrolleret og sikker måde. Du skal ikke kode noget endnu, bare overveje.

### Øvelse 3.3: Implementér tilgang til data

Du vil nu gerne implementere en ny metode i Menu-klassen, som kan bruges til at indsætte et nyt menupunkt. DesignAway-firmaet har heldigvis netop leveret en ny DCD, som kan gøre dette muligt:



Der er tilføjet en ny operation i Menu-klassen i DCD'en, nemlig *AddMenuItem()* med menupunktets titel som parameter.

Udfør følgende:

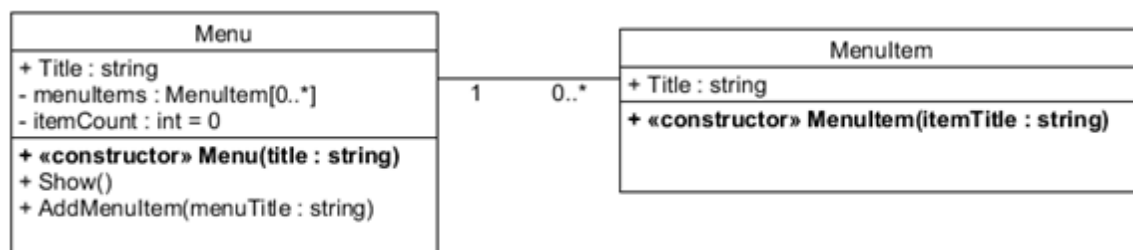
- Implementér den nye operation med en tilsvarende metode i C#. Overvej, hvordan metoden skal implementeres. Er der noget, du kan genbruge og tilrette lidt?
- Opdatér også koden i Main()-metoden, så den passer med den nye implementering.
- Kør programmet. Tilret om nødvendigt.
- Blev koden i Main()-metoden mere overskuelig? Den tidligere Main-kode indeholdt jo en del gentagelser (oprettelse af hvert menupunkt). Overvej, om der var noget fra Computational Thinking, der var i spil her.

Observer også her, at indkapslingen her opfylder sit formål, nemlig at Main-metoden ikke kan se eller påvirke den interne datastruktur, som Menu-klassen anvender til at realisere en menu.

### Øvelse 3.4: Constructors (og stereotypes)

Der er stadig lidt forbedringer, du kan lave i koden. Når du instantierer objekter for både Menu- og MenuItem-klassen, så skal du i begge tilfælde sætte titlen på begge typer objekter, *efter* du har instantieret. Det vil du gerne gøre lidt mere smart, så du faktisk sætter titlen samtidig med instantieringen.

Igen har DesignAway slået til med en ny DCD:



Notationen « ... » er det, man i UML kalder for en stereotype. En stereotype kan enten helt ændre betydningen af en konstruktion eller berige den med yderligere information, så udvikleren nemmere kan se, hvad designeren ønsker. Dvs. man kan indsætte stereotyper steder i en klasse, hvor man vil tydeliggøre noget bestemt. Der er allerede nogle få prædefinerede stereotyper, men designeren kan selv opfinde nye, som passer til konteksten. Det kommer du til at se senere.

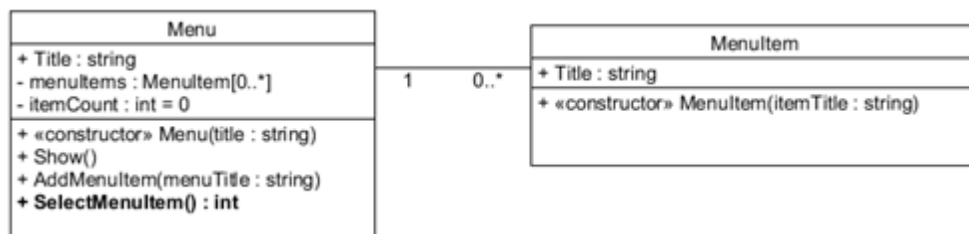
I tilfældet foroven tydeliggøres, at Menu(title : string) faktisk er en constructor for Menu-klassen, som har en parameter, nemlig titlen på menuen. Bemærk at en constructor altid vil have samme navn som klassen, her Menu, så stereotypen kunne egentlig undværes. Men man kan nemt overse dette, så stereotypen gør diagrammet mere læsbart.

Udfør følgende:

- Implementér denne DCD-ændring i C#-koden med to nye constructors, så titlen på både en menu og et menupunkt kan angives i de respektive constructors, som nu har ansvaret for at tildele værdien til Title-feltet. Husk, at constructors bliver kaldt, når du bruger new-operatoren.
- Tilret Main-koden. Overvej, om det gjorde koden lidt mere overskuelig.

### Øvelse 3.5: Vælg menupunkt

Hvad skal man med en menu, hvis man ikke også aktivt kan bruge den i sin applikation til at vælge menupunkter med, dvs. den dynamiske del af menuen. Det skal du se på nu. DesignAway er på pletten igen med en opdateret DCD:



Der er nu tilføjet en ekstra operation SelectMenuItem(), hvis eneste opgave er at returnere det valg (et heltal), som brugeren har indtastet. Bemærk, at operationen **kun skal returnere**, når et gyldigt menuvalg (i vores tilfælde 1 til 4) er foretaget, eller brugeren har valgt 0 (nul) for at forlade menu'en.

Udfør følgende:

- Implementér SelectMenuItem()-operationen i Menu-klassen i C#.

Nu har du byggestenene til at skabe og anvende et simpelt menusystem.

Udfør følgende:

- Opmatér koden i Main()-metoden, så der skrives følgende ud:
  - "Gør dit", når brugeren vælger 1.
  - "Gør dat", når brugeren vælger 2.
  - "Gør noget" når brugeren vælger 3.
  - "42", når brugeren vælger 4.
- Afprøv, og tilret.

- Kan din implementering af menusystemet håndtere menuer med 3, 7 eller 12 menupunkter? Prøv om den kan, og tilret din kode, hvis ikke, så den ikke er følsom overfor et bestemt antal menupunkter. Du må dog godt fastlægge et maksimum antal menupunkter, som aldrig må overstiges.

### Øvelse 3.6: Vis menu og valg hele tiden, indtil der afsluttes

Udfør følgende:

- Tilret Main()-metoden således, at konsolapplikationen kun afsluttes helt, når brugeren indtaster 0. Dvs. du skal sørge for, at konsolapplikationen viser menuen, håndterer valg, viser menuen igen, håndterer nyt valg, osv., indtil brugeren afslutter med menuvalget 0, hvorefter konsolapplikationen forlades helt.

### Bonusøvelse 3: Udvid menusystemet med undermenuer

Hvis du har lyst og tid, kan du overveje, hvad der skal til for at udvide dit menusystem med undermenuer til hvert menupunkt. Dette vil påvirke både præsentationen af menuen samt interaktionen med brugeren.

Undgå at starte med at kode, men lav et design først; overvej hvilke klasser der er nødvendige og hvilken adfærd, der skal til. God fornøjelse.