



Grundlæggende C#

Lab Manual

Jesper Gulmann Henriksen

03-09-2018



Indledning

Øvelsestyper

Opgaverne i disse labs er af varierende svær- og frihedsgrad. Sædvanligvis vil de indledende øvelser være relativt bundet omkring en slags "lab walkthrough", hvorimod de senere øvelser vil være friere eller være af mere eksperimenterede karakter. Derfor er de forskellige opgaver klassificeret med en indikation af type.

De indledende og grundlæggende øvelser af walkthrough-typen er ikke yderligere kategoriseret. De resterende øvelser, som man ikke nødvendigvis forventes at nå i de afsatte tid, er klassificeret som følger



Øvelser markeret med en enkelt stjerne er lidt løsere specificeret baseret på hints eller få punkter, men omhandler centrale dele af stoffet.



Øvelser markeret med to stjerner indeholder enten kun et enkelt hint, er lidt sværere eller kræver vidensindsamling.



Øvelser markeret med tre stjerner er svære og sjove udfordringer, der kan grubles over for de interesserede.

Forudsætninger

Disse opgaver forudsætter, at materialet hørende til kurset er udpakket i

C:\Teknologisk\87356

samt at Visual Studio 2017 er installeret på maskinen.

Module 1: "What is .NET and C#?"

Ingen øvelser.

Module 2: "Hello, World"

Øvelse 2.1: "Hello, World" i C#

Formålet med denne øvelse er at lave et simpelt C#-projekt i Visual Studio helt fra bunden. Derefter vil vi modificere programmet en smule og lave simpel brugerindtastning samt se, hvor man kan give command-line argumenter med til programmet og benytte disse.

Lav en ny konsol-applikation i Visual Studio

1. Start Visual Studio.
 - a. Første gang Visual Studio startes, kan I blive bedt om at vælge "Development Settings". Vælg "Visual C# Development Settings", da I så får menuer og genvejstaster, som er optimeret til udvikling i C#.
2. Vælg **File -> New -> Project...** i menuen.
3. Lav en ny C# Console Application
 - a. Project Types: **Visual C#**
 - b. Templates: **Console Application**
 - c. Name: **HelloWorld**
 - d. Location: **C:\Wincubate\87356\Module 2\Lab 2.1\Starter**
 - e. Solution Name: **HelloWorld**
 - f. Sørg for at **"Create directory for solution"** er valgt
 - g. Klik **OK**.
4. Du har nu lavet et nyt C#-projekt, som er en konsol-applikation. Check at Visual Studio som ønsket har lavet en solution bestående af ét projekt med navnet "HelloWorld" samt har lavet en C#-fil, der hedder Program.cs.
5. Luk Program.cs ved at højreklikke på dens tab i editor-vinduet eller klikke krydset i editor-vinduets øverste højre hjørne.
6. Gem alle filerne i projektet ved at vælge **File -> Save All** i menuen (**CTRL+Shift+S**).

Indsæt "Hello, World" kode og kør programmet

7. Åbn Program.cs igen ved at dobbeltklikke på den i Solution Explorer.
8. Find Main() metoden og indsæt i den tomme metode kode til at udskrive strengen Hello, World! til konsollen.
9. Når du er færdig, bør Main() metoden ligne

```
static void Main( string[] args )
{
    Console.WriteLine( "Hello, World!" );
}
```

10. Gem den aktuelle fil ved at vælge **File -> Save Program.cs** i menuen (**CTRL+S**).
11. Byg din solution ved at vælge **Build -> Build Solution** i menuen (**CTRL+Shift+B**).
12. Hvis der er kompileringsfejl skal disse rettes, hvorefter Step 11 gentages.
13. Når programmet kan kompileres uden fejl, kørs da programmet fra Visual Studio uden debugger ved at vælge **Debug -> Start Without Debugging** i menuen (**CTRL+F5**).
14. Verificér at strengen Hello World! skrives i konsol-vinduet.

15. Tryk en taste for at lukke konsol-vinduet, når der skrives "Press any key to continue" og check at applikationen lukkes ned.

Lav bruger-indtastning af navn

Vi vil nu lade brugeren indtaste sit navn og derefter skrive Hello, <<indtastet navn>> i stedet.

16. Lav en streng-variabel til at holde det indtastede navn og kald denne variabel name.
17. Skriv en sætning, der beder brugeren indtaste sit navn.
18. Skriv en sætning, der læser brugerens navn ind i name.
19. Modificér udskriftssætningen til at skrive Hello, efterfulgt af indholdet af name.
20. Når du er færdig, bør Main() metoden ligne

```
static void Main( string[] args )
{
    string name;
    Console.WriteLine( "Please enter your name: " );
    name = Console.ReadLine();
    Console.WriteLine( "Hello, {0}", name );
}
```

21. Gem alle filer.
22. Byg din solution igen og ret eventuelle kompileringsfejl.
23. Start programmet fra Visual Studio uden debugger.
24. Indtast dit navn ved promptet.
25. Verificér, at der udskrives Hello, <<indtastet navn>> som forventet.
26. Luk applikationen ned ved at trykke en taste.

Medgiv navnet til programmet som command-line argument

Vi vil nu i stedet for at indtaste brugerens navn give det med til programmet som command-line argument – først direkte gennem cmd.exe og derefter gennem Visual Studio.

27. Fjern variablen name igen.
28. Fjern sætningen, der beder brugeren om at indtaste navn.
29. Fjern sætningen, der indlæser navnet i name.
30. Erstat brugen af name i udskrivningen med det første medgivne command-line argument i args.
31. Når du er færdig, bør Main() metoden ligne

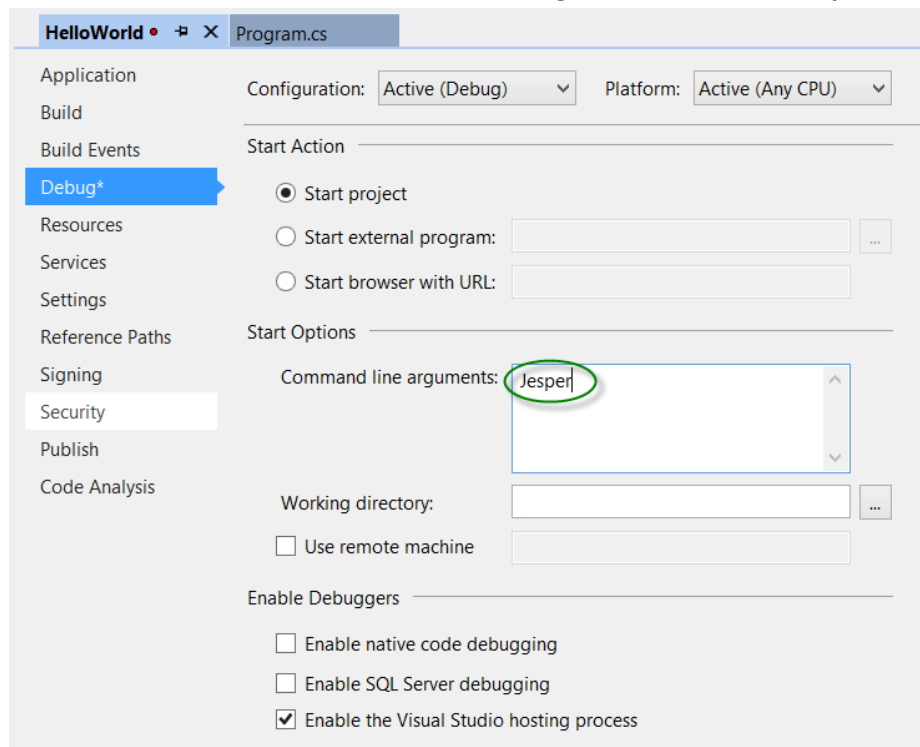
```
static void Main( string[] args )
{
    Console.WriteLine( "Hello, {0}", args[ 0 ] );
}
```

32. Byg din solution igen og ret eventuelle kompileringsfejl.
33. Kør programmet udenom Visual Studio
 - a. Start en kommando-prompt med cmd.exe eller lignende
 - i. **Bemærk:** Hvis I har Productivity Power Tools for Visual Studio installeret og aktiveret, kan I her i stedet vælge at højreklikke i Solution Explorer og vælge "Power Commands -> Open Command Prompt", hvis I foretrækker dette.

- b. Navigér til det directory, som Visual Studio har lavet til dit program:
`cd C:\Wincubate\87356\Module 2\Lab 2.1\Starter\HelloWorld`
- c. Alle filerne vedrørende HelloWorld solution findes her. Navigér nu yderligere til det subdirectory, som indeholder den exe-fil Visual Studio har lavet ud af dit C#-program:
`cd HelloWorld\bin\Debug`
- d. Skriv `dir` for at se filerne i dette directory og verificér at `HelloWorld.exe` findes her
- e. Kør programmet med dit navn som command-line argument og check, at der udskrives et korrekt Hello til dig:
`HelloWorld <<dit navn>>`

34. I stedet for at benytte denne manuelle tilgang til at teste command-line argumenter, kan man benytte Visual Studio til at hjælpe med dette, hvilket er en del mere bekvemt.

- f. Højreklik HelloWorld-projektet i Solution Explorer og vælg **Properties** i kontekstmenuen
- g. Vælg **Debug** tabben
- h. Indtast dit navn i tekstfeltet **Command line arguments** under **Start Options**, noget a la:



- i. Kør programmet igen uden debuggeren og verificér, at der udskrives Hello, `<<command-line argument>>` som forventet.
- j. Tryk en taste, så programmet lukker pænt.

35. Gentag Step 34 med din sidemands navn for at sikre, at alting er som det skal være. ☺

36. Gentag Step 34 ved at lave **command-line arguments** feltet tomt.

- k. Luk programmet ned med "Close Program" eller lignende og overvej, hvorfor dette mon skete...
- l. Vi er blevet ramt af en exception, som vi vil lære at håndtere i Module 9.
- m. Indtil vi når så langt, vil vi løse vores problem med lidt defensiv programmering ved at sikre, at der gives command-line argumenter med til programmet. Dette gøres ved at tilføje en simpel if-sætning, som checker at dette er tilfældet.

37. Skriv en if-sætning omkring indholdet i `Main()`, således at det kun udføres, hvis der er command-line argumenter til programmet; mere præcist at `args.Length` er større end 0.
38. Når du er færdig, bør `Main()` metoden ligne

```
static void Main( string[] args )
{
    if( args.Length > 0 )
    {
        Console.WriteLine( "Hello, {0}", args[ 0 ] );
    }
}
```

39. Byg din solution igen og ret eventuelle kompileringsfejl.
40. Gentag Step 34 ved at lave **command-line arguments** feltet tomt igen.
41. Kør programmet i Visual Studio uden debuggeren
- n. Verificér at problemet er løst
 - o. Luk programmet pænt ned.

Øvelse 2.2: Debugging med Visual Studio

Formålet med denne øvelse er at køre HelloWorld-programmet fra Opgave 2.1 i Visual Studio for at skabe fortrolighed med debuggeren i Visual Studio, der er en uvurdelig og avanceret hjælp i fejlfindingssituationer. Vi ser hvordan den kan bruges til at inspicere værdier af program-variabler og endda modificere disse undervejs ved stop ved breakpoints. Til sidst ser vi kort, hvordan man kan lave betingede breakpoints, der kun pauser programudførslen under særlige omstændigheder.

Kør programmet fra Opgave 2.1 under debuggeren

1. Start Visual Studio ved at dobbeltklikke på solution-filen i C:\Wincubate\87356\Module 2\Lab 2.2\Starter\HelloWorld. Alternativt kan Visual Studio åbnes og menupunktet **File -> Open -> Project/Solution...** benyttes.
2. Åbn Program.cs og verificér, at koden ligner noget, som du selv har skrevet i Opgave 2.1.
3. Sæt cursoren i den første linie med `Console.WriteLine(...)` og sæt et breakpoint med **Debug -> Toggle Breakpoint** i menuen (F9).
 - a. Dette er linie 13 i Program.cs.
 - b. Du kan hoppe direkte til linie 13 ved at vælge **Edit -> Go To...** i menuen og skrive 13 (CTRL+G).
4. Du vil se en rød markering i marginen af vinduet som tegn på, at der er sat et breakpoint i denne linie.
 - a. Du kunne alternativt have sat selvsamme breakpoint ved at klikke på det sted, hvor breakpoint indikeringen opstår
5. Kør programmet i Visual Studio med debuggeren ved at vælge **Debug -> Start Debugging** i menuen (F5).
6. Programmet vil nu køre indtil dette breakpoint nås og programudførslen vil stoppe med en gul pil, der indikerer den aktuelle sætning som er næste, som skal udføres.

Inspicér en variabel-værdi i debuggeren og single-step gennem programmet

7. Inspicér den aktuelle værdi af variablen name i debuggeren
 - a. **Bemærk:** Programmet skal på dette tidspunkt være kørende men dog pauset ved det breakpoint, som I satte ovenfor. Er programmet ikke startet og under debugging, kan Watch-vinduerne ikke ses!
 - b. Åbn Watch 1 vinduet ved at vælge **Debug -> Windows -> Watch -> Watch 1** i menuen (CTRL+D, W).
 - c. I **Name**-søjlen af Watch 1 vinduet indskriv variablen name som en variabel, der skal overvåges. Dobbeltklik for at indskrive værdien.
 - d. Variablen name overvåges nu. Check at dens initiale værdi vises som null i vinduet.
8. Udfør den næste linie i programmet ved at vælge **Debug -> Step Over** i menuen (F10).
 - e. Kontrollér at pilen, der angiver program-tælleren, er rykket til linie 14.
9. Udfør også denne linie ved at gentage Step 8.
 - f. For at kunne udføre denne linie (som jo er en `Console.ReadLine(...)`), behøver programmet dit navn som input
 - g. Indtast dit navn i konsol-vinduet
 - h. Verificér i Watch 1 vinduet, at værdien af name nu er skiftet til det indtastede navn. Den røde farve illustrerer, at denne værdi er ændret

- i. Check at programtælleren i Visual Studio nu peger på linie 15 som den næste linie, der skal udføres.
- 10. Udfør også denne linje ved at gentage Step 8.
 - j. Læg mærke til, at konsol-vinduet nu indeholder den udskrevne hilsen.
- 11. Udfør resten af programmet under debuggeren i Visual Studio ved at vælge **Debug** -> **Continue** i menuen (F5).
- 12. Læg mærke til, at når programmet køres under debuggeren lukkes det og konsol-vinduet automatisk ned, når `Main()` er færdig og venter altså ikke på tastetryk fra brugeren først.

Modificér værdier under programudførslen

- 13. Fjern det satte breakpoint i linie 13 ved at vælge **Debug** -> **Toggle Breakpoint** i menuen (F9).
 - a. Alternativt – og lettere – kunne du have fjernet dette breakpoint ved at klikken på breakpoint-indikeringen i marginen
- 14. Sæt i stedet et nyt breakpoint i linie 15, som er den linie, der skrives hilsenen ud på skærmen.
- 15. Start igen programmet under debuggeren som tidligere.
- 16. Når du har indtastet dit navn, så ændr dette i Watch 1 til din sidemands navn.
- 17. Følg i konsol-vinduet, hvad der skrives ud på skærmen i denne linie
 - b. Check at du ser din sidemands navn, selvom du oprindeligt indtastede dit eget navn
- 18. Udfør resten af programmet under debuggeren i Visual Studio.

Lav betingede breakpoints

- 19. Lav det ovenfor satte breakpoint betinget
 - a. Højreklik det og vælg **Condition...** i menuen
 - b. Check at **Condition** samt **Is True** er valgt
 - c. Indskriv
 - `name == "Rafael"`
 - som betingelse
 - d. Klik **Close**.
- 20. Kør programmet under debuggeren
 - e. Indskriv dit eget navn
 - f. Check at programmet ikke stopper udførslen ved det satte breakpoint.
- 21. Kør programmet under debuggeren igen
 - g. Indskriv Rafael
 - h. Check at programmet rent faktisk stopper ved det satte breakpoint
 - i. Udfør resten af programmet.

Breakpoints er kun aktive under debugging

- 22. Kør til sidst programmet (stadig med ovenstående breakpoint) uden debugging.
 - a. Check at programmet ikke stoppes ved dette – eller noget som helst andet – breakpoint uanset den indtastede værdi.

Øvelse 2.3: Formateringer af Output (★)

Formålet med denne øvelse er at undersøge formateringerne af tal, når de udskrives med `Console.WriteLine()`.

Indtast et heltal og formatér det

1. Lav et projekt i Visual Studio til ny C# konsol-applikation
 - a. Kald projektet "Formatting" og
 - b. Placer det i
C:\Wincubate\87356\Module 2\Lab 2.3\Starter
2. I `Program.cs` skal `Main()` skrives, så
 - a. der indtastes et tal ind i en streng-variabel på samme måde, som jeres navne blev indtastet i Øvelse 2.1.
 - b. Den indtastede streng skal konverteres til en integer. Dette kan gøres med f.eks.
`int i = int.Parse(number);`
hvis det forudsættes, at `number` indeholder den indtastede streng
 - i. **Ignorér her de runtime-fejl, som kan opstå ved indtastning af tal, der ikke svarer til korrekte heltal!**
 - c. Det konverterede heltal skal efterfølgende udskrives som
 - i. En integer ('d')
 - ii. Et tal ('n')
 - iii. En valuta-enhed ('c')
3. Byg programmet og korriger eventuelle fejl.
4. Kør programmet og observer forskellene på de tre måder at formatere heltallet på
 - a. Hvad er forskellene?
 - b. Prøv både at indtaste tal mindre og større end 1000, hhv.
5. Justér på Regional Settings i det underliggende operativsystem
 - a. Hvis den nuværende opsætning er da-DK, så skift til en-US
 - b. Hvis den nuværende opsætning er en-US, så skift til da-DK
6. Gentag Step 4.
 - a. Hvilken forskel gjorde ændringen i Regional Settings?
7. Skift Regional Settings tilbage til de oprindeligt konfigurerede værdier.

Module 3: "Value Types and Expressions"

Øvelse 3.1: ".NET Doktor"

I denne øvelse vil vi skrive et program, der kan beregne vores promille på baggrund af, hvor mange genstande vi har drukket, så vi kan tage dette program med den næste firmafest eller julefrokost. Sundhedsstyrelsens formel for beregning af promille er givet ved

$$\text{promille} = \frac{12 * \text{antal genstande}}{k * \text{vægt}}$$

hvor k er en fast faktor, der er 0.68 for mænd og 0.55 for kvinder. Vi vil indtaste køn og vægt via konsol-vinduet og lade programmet beregne vores promille.

Formålet med denne øvelse er at lære at lave beregningsudtryk og variabler. Derudover skabes lidt mere erfaring med gængs bruger-input og -output via konsol-vinduet.

Beregn promille

1. Lav et projekt i Visual Studio til ny C# konsol-applikation
 - a. Kald projektet "DotNetDoktor" og
 - b. Placér det i
C:\Wincubate\87356\Module 3\Lab 3.1\Starter
2. I Program.cs skal Main() skrives som følger
 - a. Udskriv på skærmen linien "Indtast dit køn ('m'/'k'):"
 - b. Lav en streng-variabel s , og indlæs en linie fra input ind i s
 - c. Erklær en variabel k af typen double
 - d. Hvis det første indtastede tegn er et 'm', så sæt k til mændenes værdi. Ellers sæt k til kvindernes værdi. Brug en if-sætning til formålet.
 - i. **Ignorér her og i hele resten af opgaven de runtime-fejl, som kan opstå ved forkerte indtastninger af strenge og tal!**
 - e. På samme måde udskrives "Indtast din vægt [kg]: ", og der indlæses en streng ind i en variabel v
 - f. Indholdet af v skal konverteres til en variabel vægt af type double. Brug sætningen
double vægt = double.Parse(v);
 - g. På samme måde igen udskrives "Indtast antal genstande: ", og der indlæses en streng ind i en variabel g
 - h. Indholdet af g skal denne gang konverteres til en variabel genstande af type int
 - i. Endelig kan erklæres en variabel promille af type double, der indeholder den beregnede promille efter ovennævnte formel
 - j. Udskriv til sidst "Din promille er " efterfulgt af den beregnede promille udskrevet med to decimalers nøjagtighed

3. Når du er færdig, bør Main() metoden ligne

```
static void Main( string[] args )
{
    Console.WriteLine( "Indtast dit køn ('m'/'k'): " );
    string s = Console.ReadLine();

    double k;
    if( s[ 0 ] == 'm' )
    {
        k = 0.68;
    }
    else
    {
        k = 0.55;
    }

    Console.WriteLine( "Indtast din vægt [kg]: " );
    string v = Console.ReadLine();
    double vægt = double.Parse( v );

    Console.WriteLine( "Indtast antal genstande: " );
    string g = Console.ReadLine();
    int genstande = int.Parse( g );

    double promille = ( 12 * genstande ) / ( k * vægt );
    Console.WriteLine( "Din promille er {0:f2}", promille );
}
```

4. Byg programmet og korriger eventuelle fejl.
5. Kør programmet og beregn jeres promille ved sidste julefrokost ☺.

Øvelse 3.2: "Definér en type for spillekort" (★)

I denne øvelse vil vi definere og bruge en værditype, der repræsenterer et enkelt kort, som vi kender det fra et gængs kortspil.







Definér kort-type

1. Lav et projekt i Visual Studio til ny C# konsol-applikation
 - a. Kald projektet "CardTest" og
 - b. Placer det i
C:\Wincubate\87356\Module 3\Lab 3.2\Starter
2. Lav en ny C#-kodefil ved at højreklikke på CardTest projektet i Solution Explorer og vælge **Add** -> **Class...** i menuen
 - a. Name: Card.cs
3. Da vi ikke skal lave en klasse, skal

```
class Card
{
}
```

slettes først.
4. Inde i namespace't i Card.cs skal der defineres en enum type, der angiver en kulør.
 - a. Kald denne type Suit
5. På samme måde laves der en enum type, der angiver kortets værdi, f.eks. Two, Ten, eller Jack.
 - a. Kald denne type Rank
6. Benyt derefter Suit og Rank til at definere en struct type, Card, der angiver selve kortet.
7. Byg programmet og korriger eventuelle fejl.

Lav kort-variabler

8. Lav nu i Main() to lokale variabler, card1 og card2, af typen Card.
9. Initialiser dem til jeres favorit-starthånd i Hold 'Em Poker, eksempelvis
 - a.   eller
 
 - b.  
10. Prøv at udskrive variablerne med Console.WriteLine(). Hvad sker der?

Øvelse 3.3: ".NET Doktor – Nu med promille advarsel" (★)

Denne øvelse udgør en fortsættelse af promille-programmet i Øvelse 3.1.

Vi udvider programmet, så der giver en advarsel i tilfælde af, at den aktuelle promille er for høj til bilkørsel hjem. Denne opgave illustrerer kommentarer, brug af Visual Studios hjælpefunktioner samt farver i konsol-vinduet, og der skrives et par kommentarer.

Advar om for høj promille

1. Udvid programmet med en if-sætning en advarsel ved at gøre skriftfarven i konsol-vinduet rød under udskrift af promillen, såfremt promillen er højere end de tilladte 0.5 til bilkørsel.
 - a. Den relevante property, der skal sættes er `Console.ForegroundColor`
 - b. Brug Visual Studio til at finde ud af, hvordan det skal gøres
 - i. Forslag 1: Brug **Help** -> **Index** i menuen
 - ii. Forslag 2: Skriv
`Console.ForegroundColor`
i editor-vinduet på stedet, hvor koden skal være, stil cursoren i "ForegroundColor" og tryk F1
 - iii. Forslag 3: Lad IntelliSense (og kraften) guide jer..!
 - c. I kan nu skrive den nødvendige if-sætning
2. Kør programmet og afgiv inputs, som giver en promille under og over, hhv., de 0.5 og test, at dit program korrekt advarer om promillen.
3. Hvis det skal være rigtigt pænt, så sørg for at sætte den eksisterende skriftfarve tilbage, når I har udskrevet promillen. Hvordan gøres dette?

Kommentér programmet

4. Kommentér nu dit program ved at indsætte nedenstående kommentarer på de relevante steder i koden
 - a. "Indtastning af køn"
 - b. "Indtastning af vægt"
 - c. "Indtastning af antal genstande"
 - d. "Beregning af promille"
5. Test at dit program stadig kan kompilere.

Øvelse 3.4: "Udforsk DateTime" (☆☆)

I denne øvelse vil vi bruge IntelliSense og Visual Studios hjælpesystem til udforske den indbyggede DateTime type, der – på trods af at den ikke er belønnet med sit eget keyword i C# – er en særdeles nyttig værdi-type i mange sammenhænge.

Mål programmets udførselstid

- Undersøg hvordan DateTime hænger sammen med TimeSpan typen.
- Åbn dit promille-program fra Øvelse 3.1 eller 3.3 og brug TimeSpan til at måle hvor lang tid, det tager programmet at køre og udskriv dette inden programmet lukker ned.
- Programmet skal udskrive starttidspunktet i et specielt format givet ved
Programmet startede 2015-12-24 22:17:55.164
- Programmet skal derefter udskrive den forløbne tid i millisekunder, f.eks.
Programmet kørte 5447 millisekunder

Module 4: "Reference Types and Statements"

Øvelse 4.1: "Arrays, spillekort og iterationer"

Vi bruger i denne opgave typen Card fra Øvelse 3.2 og laver et helt sæt spillekort for at eksperimentere med arrays. Vi vil iterere gennem sættet af kort med de forskellige konstruktioner for iteration, samt bruge if- og switch-sætninger til at behandle og udskrive kortene.

Initialiser kortspillet

1. Åbn det eksisterende Decktest projekt i Visual Studio
 - a. Det findes i
C:\Wincubate\87356\Module 4\Lab 4.1\Starter
 - b. Check implementationen af typen Card fra Øvelse 3.2
 - i. Hvis du ønsker at benytte din egen implementation af Card, kan du erstatte den eksisterende implementation i Card.cs.
2. Find "TODO 1" i koden
3. Erklær og instantiér en variabel deck til at indeholde et array af Card af længde 52
4. Når du er færdig, bør resultatet ligne

```
// TODO 1  
Card[] deck = new Card[ 52 ];
```

5. Kortspillet skal nu sættes til at indeholde et komplet sæt kort. Find "TODO 2" i koden.
 - a. **Bemærk:** Denne findes i en såkaldt region, der defineres med
#region XXX ... #endregion
i kodefilen. Disse regioner kan åbnes og kollapses ved klik på firkanten ved regionens begyndelse yderst til venstre i editor-vinduet i Visual Studio
 - b. Gør "TODO 2" synlig, såfremt den tilhørende region er kollapsed.
6. Erklær og instantiér en integer-variabel indexToFill til at indeholde 0. Denne bruges til at iterere gennem deck.
7. Lav to foreach-sætninger inde i hinanden, der løber gennem alle kombinationer af værdierne, som Suit og Rank kan antage.
 - a. **Bemærk:** Brug ikke var i foreach-sætningerne! (Dette skyldes den teknikalitet, at Enum.GetValues() nedenfor returnerer et array af object).
 - b. Som beskrevet i [Troelsen] kan man få et array af alle værdierne i enumerationerne vha. udtrykkene
Enum.GetValues(typeof(Suit))
Enum.GetValues(typeof(Rank))
8. Sæt suit og rank (for det kort i deck som indiceres af indexToFill) til at være de ovenfor valgte i foreach sætningerne.
9. Tæl indexToFill én op.



10. Når du er færdig, bør resultatet ligne

```
// TODO 2
int indexToFill = 0;
foreach( Suit suit in Enum.GetValues( typeof( Suit ) ) )
{
    foreach( Rank rank in Enum.GetValues( typeof( Rank ) ) )
    {
        deck[ indexToFill ].suit = suit;
        deck[ indexToFill ].rank = rank;
        indexToFill++;
    }
}
```

11. Byg programmet og korriger eventuelle fejl.

Udskriv kortspillet

Vi vil nu lave et stykke kode, der udskriver hvert kort i deck med en to-bogstavs streng med rangen først. For kuløren benyttes 'c', 'd', 'h' og 's' for klør, ruder, hjerter og spar, henholdsvis. For rang benyttes '2', '3', ..., '9', 'T', 'J', 'Q', 'K', 'A' for værdierne fra 2 op til es, henholdsvis.

Eksempelvis skal  udskrives som "Qh" og  som "2s" osv.

12. Find "TODO 3" i koden.
13. Lav en foreach-sætning der løber gennem alle kort i deck med variablen card af type Card. For hver iteration skal følgende gøres:
14. Erklær en tegn-variabel rankOutput. Denne vil komme til at indeholde det tegn som udgør rangen af card.
15. Brug en switch-sætning på card.rank og sæt rankOutput til den tilsvarende tegn-værdi
 - a. Lad Visual Studio auto-generere skabelonen til switch'en
 - i. Skriv switch og tryk TAB to gange
 - ii. Skriv card.rank inde i switch-parantesen og tryk derefter TAB og pil ned.
 - b. Brug lidt copy-paste til at generere resten. 😊
16. Erklær en tegn-variabel suitOutput. Denne vil komme til at indeholde det tegn som udgør kuløren af card.
17. Brug en switch-sætning på card.suit og sæt suitOutput til den tilsvarende tegn-værdi
18. Udskriv vha. Console to-bogstavs strengen for card efterfulgt af et mellemrum.
19. Når du er færdig, bør resultatet ligne

```

// TODO 3
foreach( Card card in deck )
{
    char rankOutput;
    switch( card.rank )
    {
        case Rank.Two:
            rankOutput = '2';
            break;
        case Rank.Three:
            rankOutput = '3';
            break;
        case Rank.Four:
            rankOutput = '4';
            break;
        case Rank.Five:
            rankOutput = '5';
            break;
        case Rank.Six:
            rankOutput = '6';
            break;
        case Rank.Seven:
            rankOutput = '7';
            break;
        case Rank.Eight:
            rankOutput = '8';
            break;
        case Rank.Nine:
            rankOutput = '9';
            break;
        case Rank.Ten:
            rankOutput = 'T';
            break;
        case Rank.Jack:
            rankOutput = 'J';
            break;
        case Rank.Queen:
            rankOutput = 'Q';
            break;
        case Rank.King:
            rankOutput = 'K';
            break;
        case Rank.Ace:
            rankOutput = 'A';
            break;
        default:
            rankOutput = '?';
            break;
    }
}

```

```

char suitOutput;
switch( card.suit )
{
    case Suit.Clubs:
        suitOutput = 'c';
        break;
    case Suit.Diamonds:
        suitOutput = 'd';
        break;
    case Suit.Hearts:
        suitOutput = 'h';
        break;
    case Suit.Spades:
        suitOutput = 's';
        break;
    default:
        suitOutput = '?';
        break;
}

Console.Write( "{0}{1} ", rankOutput, suitOutput );
}

```

Bemærk: En alternativ løsning til 4) ville være at behandle tilfældene *Rank .Two* til *Rank .Nine* under ét ved at benytte et lidt mere avanceret trick, idet alle værdierne fra 2 til og med 9 kan håndteres fælles med sætningen

```
rankOutput = (char)( '2' + ( (char) card.rank ) );
```

Denne løsning er implementeret i den medfølgende opgaveløsning under "Solution".

20. Byg programmet og korriger eventuelle fejl.
21. Kør programmet og verificér, at alle 52 spillekort udskrives korrekt.

Bland kortene

Vi vil nu blande kortene i sættet. En effektiv blandingsalgoritme – der dog ikke er let at lave med et fysisk spil kort ☺ – er at løbe gennem sættet og bytte hvert af kortene ud med et tilfældigt valgt kort fra resten af sættet.

22. Find "TODO 4" i koden.
23. Først erklæres og initialiseres en generator af tilfældige tal, der i .NET er typen `Random`. Kald variablen `random`.
24. Lav dernæst en for-løkke med en integer tælle-variabel kaldet `i`, der løber gennem `deck`.
25. Inde i løkken initialiseres variablen `j` til at indeholde et tilfældigt tal mellem 0 og længden af `deck` - 1.
 - a. Dette gøres vha. `Next()`-metoden på `Random` som følger


```
int j = random.Next( deck.Length );
```
26. Byt dernæst det `Card`, der findes i `deck` ved indeks `i` ud med det ved indeks `j`.
27. Når du er færdig, bør resultatet ligne


```
// TODO 4
Random random = new Random();
for( int i = 0 ; i < deck.Length ; i++ )
{
    int j = random.Next( deck.Length );
    Card temp = deck[ i ];
    deck[ i ] = deck[ j ];
    deck[ j ] = temp;
}
```

28. Byg programmet og korriger eventuelle fejl.
29. Kør programmet og verificér, at sættet med spillekort nu er blandet tilfældigt.

Find Spar To

Vi vil nu benytte én af de indbyggede Array-metoder, `Array.IndexOf()`, til at lokalisere et specifikt kort i det blandede sæt kort.

30. Find "TODO 5" i koden.
31. Lav en variabel `deuceOfSpades` af typen `Card`.
32. Sæt den til at være ²♠.
33. Find `deuceOfSpades`'s indeks i `deck` vha. `Array.IndexOf()`.
34. Udskriv til sidst "2s er på position " efterfulgt af den beregnede position.
35. Når du er færdig, bør resultatet ligne

```
// TODO 5
Card deuceOfSpades;
deuceOfSpades.suit = Suit.Spades;
deuceOfSpades.rank = Rank.Two;
Console.WriteLine( "2s er på position {0}",
    Array.IndexOf( deck, deuceOfSpades ) );
```

36. Byg programmet og korriger eventuelle fejl.
37. Kør programmet og verificér, at sættet med spillekort nu er blandet tilfældigt og at ²♠ er at finde på den position, der udskrives.

Find par blandt spillekortene

Til sidst vil vi lede efter par blandt de blandede kort i `deck`. For at begrænse resultaterne en smule vil vi kun undersøge de første antal kort, hvor dette antal indtastes via konsol-vinduet.

38. Find "TODO 6" i koden.
39. Lav en for-løkke med en integer tælle-variabel kaldet `i`, der løber gennem `deck`.
40. Lav indeni denne løkke en ny for-løkke med en integer tælle-variabel kaldet `j`, der løber gennem `deck`, men starter fra indeks `i + 1` for ikke at tælle parrene med flere gange.
41. Sammenlign inde i den inderste for-løkke nu kortene på position `i` og `j` og udskriv nu
Position `i` og `j` udgør et par
såfremt der er tale om et par på de to positioner.

42. Når du er færdig, bør resultatet ligne

```
// TODO 6
Console.WriteLine( "Indtast antal kort, der skal undersøges: " );
int number = int.Parse( Console.ReadLine() );

if( number > 0 )
{
    for( int i = 0 ; i < number ; i++ )
    {
        for( int j = i + 1; j < number ; j++ )
        {
            if( deck[ i ].rank == deck[ j ].rank )
            {
                Console.WriteLine( "Position {0} og {1} udgør et par", i, j );
            }
        }
    }
}
```

43. Byg programmet og korriger eventuelle fejl.

44. Kør programmet med forskellige indtastede værdier og verificér, at de beregnede par passer med det udskrevne sæt kort samt det indtastede antal.

Øvelse 4.2: "Henfald af promille" (★)

Denne øvelse udgør en fortsættelse af promille-programmet i Øvelse 3.3.

For at kunne estimere hvornår vi igen kan hente bilen og er ovenpå igen efter indtagelsen af de mange genstande, vil vi modificere løsningen fra Øvelse 3.3 til at udskrive "henfaldet" af beruselsen.

Det anslås approksimativt, at promillen pr. time (indtil den når 0.0 eller under) aftager med

$$\text{forbrænding} = \frac{c}{k}$$

hvor c er en fast faktor, der er 0.151 for mænd og 0.097 for kvinder.

Promillen en time efter indtagelse af genstandene er altså givet som *original promille - forbrænding*, osv.

Udskriv henfaldet af beruselsen

- Åbn det eksisterende DotNetDoktor projekt i Visual Studio fra
C:\Wincubate\87356\Module 4\Lab 4.2\Starter
- Modificér programmet til for hver time at udskrive teksten
Din promille efter t timer er *promille*
indtil promillen beregnes til at være 0.0 (eller under).
- Hvis der er tid, så skriv – som i Øvelse 3.3 – ovenstående tekst med rødt, såfremt promillen stadig er over de tilladte 0.5 for at køre bil.

Øvelse 4.3: "Flere-dimensionelle arrays" (☆☆)

Vi vil i denne opgave konstruere initialiseringsudtryk for både to- og tre-dimensionelle arrays samt gennemløbe dem.

Lav to-dimensionelt array af strenge

- Konstruér et 2 x 3 array af strenge med følgende indhold

Thebe	Io	Ananke
Ganymedes	Leda	Sinope

- Udskriv indholdet med en foreach-løkke.

Lav tre-dimensionelt array af heltal

- Lav et 2 x 3 x 4 array af heltal og fyld det med tallene fra 0 til 23.
- Udskriv indholdet med en foreach-løkke.

Øvelse 4.4: "Sortering af arrays" (☆☆)

Vi vil i denne undersøgelse sorteringer af arrays af Card.

Sortér arrays af heltal

- Åbn det eksisterende SortingTest projekt i Visual Studio i
C:\Wincubate\87356\Module 4\Lab 4.4\Starter
der er baseret på løsningen fra Øvelse 4.1. Løsningen indeholder to regioner med
 - Et integer-array numbers
 - Et Card-array deck .
- Kør programmet og se, at en numbers udskrives usorteret.
- Lokalisér "TODO 1: Sort numbers" og indsæt kode til at sortere numbers via System.Array metoder.
- Byg og kør dit program.
- Kontrollér at numbers nu udskrives i sorteret rækkefølge.

Sortér arrays af Card

- Fjern regionen numbers fra koden.
- Indkommentér koden i regionen deck.
- Lokalisér "TODO 2: Sort deck" og indsæt kode til at sortere kortene i deck via System.Array metoder.
- Byg og kør dit program.
- Hvad sker der nu? Hvorfor...?
- Lokalisér "TODO 3: Custom sort algorithm" og indsæt kode til at sortere kortene i deck på rangen af kortene.
 - Lav sorteringen helt fra bunden ved at kode din favorit-sorteringsalgoritme på dette sted
 - Eksempelvis BubbleSort eller lignende...
- Byg og kør dit program.
- Kontrollér, at deck nu udskrives i rækkefølge sorteret på kortenes værdi.

Bare rolig...

- De senere kapitler giver den elegante løsning på dette problem! 😊

Module 5: "Methods"

Øvelse 5.1: "Find vokaler i strenge"

I denne øvelse vil vi lave to metoder, hvor den ene kalder den anden, således at vi tæller antallet af vokaler i en streng, som brugeren indtaster.

Lav IsVowel() metoden

1. Lav et projekt i Visual Studio til ny C# konsol-applikation
 - a. Kald projektet "FindVowels" og
 - b. Placer det i
C:\Wincubate\87356\Module 5\Lab 5.1\Starter
2. I Program.cs på linie med Main() metoden defineres en metode ved navn IsVowel
 - a. Metoden skal acceptere et argument c af typen char og returnere typen bool
 - b. Metoden skal være static som Main() af grunde, som vi belyser i Module 7
3. Kroppen af IsVowel() skal nu udfyldes således, at der returneres true, hvis c er en vokal og false ellers
 - a. Konvertér først c til lower-case. Dette kan gøres vha.
char.ToLower(c)
 - b. Lav dernæst en switch-sætning på den konverterede tegnværdi og lav cases for alle vokalerne.
 - i. Returnér true for disse cases
 - ii. Returnér false ellers
4. Når du er færdig, bør resultatet ligne

```
static bool IsVowel(char c)
{
    switch( char.ToLower( c ) )
    {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
        case 'y':
        case 'æ':
        case 'ø':
        case 'å':
            return true;
        default:
            return false;
    }
}
```

5. Byg programmet og korriger eventuelle fejl.

Lav FindVowels() metoden

6. I Program.cs på linie med Main() metoden defineres en metode ved navn FindVowels
 - a. Metoden skal acceptere et argument s af typen string og returnere typen int

- b. Metoden skal ligeledes være `static` som `Main()`
- 7. Kroppen af `FindVowels()` skal nu udfyldes således, at der returneres antallet af vokaler i strengen `s`
 - c. Lav en lokal variabel `count` og initialiser den til 0
 - d. Benyt en `foreach`-sætning til at iterere gennem alle tegn `c` i `s`
 - e. Inde i loopet kaldes `IsVowel()` på `c` og `count` tælles én op, hvis denne metode returnerer `true`
 - f. Slutteligt returneres `count` nederst i metoden.
- 8. Når du er færdig, bør resultatet ligne

```
static int FindVowels(string s)
{
    int count = 0;

    foreach (char c in s)
    {
        if (IsVowel(c))
        {
            count++;
        }
    }

    return count;
}
```

- 9. Byg programmet og korriger eventuelle fejl.

Lav `Main()` metoden

- 10. I `Program.cs` skrives `Main()` metoden således, at der indtastes en streng via `Console`, som `FindVowels()` metoden kaldes på, hvorefter resultatet udskrives
 - a. Udskriv
Indtast en streng:
på skærmen
 - b. Erklær en lokal variabel `input` og indlæs den indtastede streng ind i denne
 - c. Erklær en lokal variabel `vowels` og sæt den til at indeholde returværdien af `FindVowels()` kaldt på `input`
 - d. Udskriv
Der er `vowels` vokaler i strengen `"input"`
med de korrekte værdier.
 - e. **Bemærk:** Husk anførelstegnene ovenfor!
- 11. Når du er færdig, bør resultatet ligne


```

static void Main(string[] args)
{
    Console.WriteLine( "Indtast en streng: ");
    string input = Console.ReadLine();

    int vowels = FindVowels( input );
    Console.WriteLine( "Der er {0} vokaler i strengen \"{1}\"",
        vowels, input );
}

```

12. Byg programmet og korriger eventuelle fejl.
13. Kør programmet og afprøv det på strengene
 - Rød grød med fløde
 - Rød grød med mere FLØDE
 - Onomatopoietikon
 - ...

Øvelse 5.2: "Refaktorér til metoder" (★)

Denne øvelse viser, hvorledes man kan udtrække metoder fra eksisterende kode. Vi vil i koden fra løsningen til Øvelse 4.1 udtrække metoder for de centrale kode-blokke.

Lav FillDeck() metode manuelt

1. Åbn det eksisterende DeckTest projekt i Visual Studio i
C:\Wincubate\87356\Module 5\Lab 5.2\Starter
der er baseret på løsningen fra Øvelse 4.1. Løsningen indeholder et antal regioner, der er oplagte kandidater til at blive separate metoder i Program
2. Lav FillDeck() metode på linie med Main()
 - a. Lokalisér "TODO 1: Create FillDeck() method" i koden
 - b. Metoden skal acceptere et argument deck af typen Card[] og returnere void
 - c. Metoden skal ligeledes være static som Main()
 - d. Flyt indholdet af region Fill deck of cards til metode-kroppen af FillDeck()
 - e. Slet #region Fill deck of cards ... #endregion og erstat koden med et kald af FillDeck() metoden med deck
3. Når du er færdig, bør resultatet ligne

```
static void Main( string[] args )
{
    ...

    // TODO 1: Create FillDeck() method
    FillDeck(deck);

    ...
}

static void FillDeck(Card[] deck)
{
    int indexToFill = 0;
    foreach (Suit suit in Enum.GetValues(typeof(Suit)))
    {
        foreach (Rank rank in Enum.GetValues(typeof(Rank)))
        {
            deck[indexToFill].suit = suit;
            deck[indexToFill].rank = rank;
            indexToFill++;
        }
    }
}
```

4. Byg programmet og korriger eventuelle fejl.
5. Kør programmet og verificér, at programmet stadig virker på samme måde som før.

Lav ShuffleDeck() metode automatisk

6. Lav ShuffleDeck() metode på linie med Main()
 - a. Lokalisér "TODO 2: Create ShuffleDeck() method" i koden

- b. Markér al kode i regionen, dvs. fra
Random random...
til
}
c. Højre-klik den markerede kode i Visual Studio og vælg **Quick Actions...** -> **Extract Method** i menuen.
d. Angiv
ShuffleDeck
som metodenavn
e. Klik "Apply".
f. Slet #region Shuffle deck ... #endregion
7. Når du er færdig, bør resultatet ligne

```
static void Main( string[] args )
{
    ...

    // TODO 1: Create FillDeck() method
    FillDeck(deck);

    // TODO 2: Create ShuffleDeck() method
    ShuffleDeck(deck);

    ...
}

private static void ShuffleDeck(Card[] deck)
{
    Random random = new Random();
    for (int i = 0; i < deck.Length; i++)
    {
        int j = random.Next(deck.Length);
        Card temp = deck[i];
        deck[i] = deck[j];
        deck[j] = temp;
    }
}
```

8. Byg programmet og korriger eventuelle fejl.
9. Kør programmet og verificér, at programmet stadig virker på samme måde som før.

Lav de resterende metoder

Brug valgfrit enten den manuelle eller den automatisk fremgangsmåde til at udtrække de resterende tre metoder:

- OutputDeck()
- FindCard()
 - Metode-kroppen skal kun bestå af den ene linie med Console.WriteLine()
 - Rename bagefter deuceOfSpades til noget mere sigende i metode-headeren

- g. Dette kan gøre ved at vælge **Edit** -> **Refactor** -> **Rename...** i menuen (eller via højrekliksmenuen)
- **FindPairs()**
 - h. Metode-kroppen skal indeholde fra
 `if(number > 0)`
til
 `}`

Spørgsmål

- Hvordan håndteres yderligere parametre i de to forskellige fremgangsmåder?
 - Hvad sker der ved `FindCard()`?
 - Hvad sker der ved `FindPairs()`?
- Hvilken metode foretrækker du?

Øvelse 5.3: "Metoder med optional og named parametre" (★)

Formålet med denne øvelse er på et simpelt eksempel at vise, hvorledes optional og named parametre kan benyttes i C#. Vi vil lave et simpelt veksleprogram, hvor et beløb i en givet valuta kan indtastes sammen med en vekslekurs for denne (eks. EUR eller USD), hvorefter programmet beregner det tilsvarende beløb i DKK.

Lav ComputeCurrency() metode

1. Lav et projekt i Visual Studio til ny C# konsol-applikation
 - a. Kald projektet "Currency" og
 - b. Placer det i
C:\Wincubate\87356\Module 5\Lab 5.3\Starter
2. I Program.cs på linie med Main() metoden defineres en metode ved navn ComputeCurrency
 - a. Metoden skal acceptere et første argument amount af typen double
 - b. Metoden skal acceptere et andet argument rate af typen double
 - c. Metoden skal returnere en værdi af typen double
 - d. Metoden skal være static som Main() af grunde, som vi belyser i Module 6.
3. Kroppen af ComputeCurrency() skal nu udfyldes således, at der returneres værdien af beløbet i amount konverteret med en valutakurs svarende til værdien af rate.
4. Når du er færdig, bør resultatet ligne

```
static double ComputeCurrency( double amount, double rate )
{
    return amount * rate;
}
```

5. Lav kode i Main() til at teste, hvad USD 50 svarer til i DKK ved en vekslekurs på 5,7781.
6. Når du er færdig, bør resultatet ligne

```
static void Main( string[] args )
{
    Console.WriteLine( ComputeCurrency( 50, 5.7781 ) );
}
```

7. Kør programmet og test at din løsning er korrekt.

Lav parametrene i ComputeCurrency() optional

8. Modificér ComputeCurrency(), således at
 - a. amount er optional med en default-værdien værende 100,00, som sædvanligvis er standardbeløbet man sammenligner valutaer i.
 - b. rate er optional med en default-værdien værende 7,4520, der er raten for EUR, som vi vil lade være default-valuaten.
9. Når du er færdig, bør resultatet ligne

```
static double ComputeCurrency( double amount = 100.00,
                                double rate = 7.4520 )
{
    return amount * rate;
}
```

10. Lav den letteste og pæneste kode i Main() til at teste,

- c. hvad 200 GBP til kurs 8,9011 svarer til i DKK.
- d. hvad 50 enheder i default-valutaen svarer til i DKK.
- e. hvad et default antal enheder i default-valutaen svarer til i DKK.

11. Når du er færdig, bør resultatet ligne

```
static void Main( string[] args )
{
    Console.WriteLine( ComputeCurrency( 200, 8.9011 ) );
    Console.WriteLine( ComputeCurrency( 50 ) );
    Console.WriteLine( ComputeCurrency() );
}
```

12. Kør programmet og test at din løsning er korrekt.

Brug named parametre i ComputeCurrency()

13. Lav nu den letteste og pæneste kode i Main() til at teste,

- a. Hvad et default antal enheder i GBP til kurs 8,9011 svarer til i DKK.

14. Når du er færdig, bør resultatet ligne

```
static void Main( string[] args )
{
    Console.WriteLine( ComputeCurrency( rate: 8.9011 ) );
}
```

15. Kør programmet og test at din løsning er korrekt.

Hvis du har lyst...

... så kan du lave ComputeCurrency() metoden om til at returnere en streng indeholdende en korrekt formatteret valuta-streng i stedet for at returnere en double med beløbet.

- Brug string.Format med "c" til at lave en korrekt udskrivning af det korrekte beløb med valuta-indikation og to decimaler.

Øvelse 5.4: "Metoder med reference-parametre" (★)

Formålet med denne øvelse er at illustrere, hvorledes reference-parametre virker i C#.

Lav Swap() metode

1. Åbn det eksisterende SwapTest projekt i Visual Studio i
C:\Wincubate\87356\Module 5\Lab 5.4\Starter
som indeholder kode i Main(), der tester en endnu ikke defineret metode kaldet Swap().
2. Udfyld kroppen af Swap(), så den bytter værdierne af i og j om.
3. Hvad sker der, når du kører programmet? Hvorfor?

Modificér Swap() metode

4. Modificér programmet, så Swap() rent faktisk lever op til sit navn
 - b. Hvad skal der ændres?
5. Kør programmet og test at din løsning er korrekt.

Lav en Vat() metode

6. Tilføj til programmet en metode Vat(), der accepterer et array af tal og lægger 25% til hver af indgangene
 - a. Ligesom i Swap() med reference modifier skal effekten af metoden været bevaret selv efter kaldet af Vat()
 - b. Hvilken modifier skal Vat() mon have? Hvorfor?
 - c. Kan foreach bruges? Hvorfor (ikke)?
 - d. **Bemærk:** Der skal ikke bruges params i denne opgave! 😊
7. Kør programmet og test at din løsning opfylder kravene.

Øvelse 5.5: "Average() med variabelt antal argumenter" (★)

Denne øvelse viser, hvorledes man kan lave metoder med et variabelt antal argumenter vha. params i C#.

Lav Average() metode

- Lav et projekt i Visual Studio til ny C# konsol-applikation kaldet "ComputeAverage"
 - Placer det i
C:\Wincubate\87356\Module 5\Lab 5.5\Starter
- Lav en metode Average() i Program-klassen med egenskaberne
 - Average() skal kunne kaldes med et vilkårligt antal argumenter
 - Average(1.76, 3.14) returnerer 2,45
 - Average(42, 87, 112, 176) returnerer 104,25
 - Average() returnerer 0
- Test din metode fra Main() med følgende kode

```
static void Main( string[] args )
{
    Console.WriteLine( Average( 42, 87, 112, 176 ) );
    Console.WriteLine( Average( 1.76, 3.14 ) );
    Console.WriteLine( Average() );
}
```


Øvelse 5.6: "Fibonacci-tal" (☆☆)

Denne øvelse beskæftiger sig med rekursive metoder, der er gode løsninger til induktivt definerede problemer.

Fibonacci-tallene er en talrække, hvor hvert element er defineret ud fra de to foregående elementer i rækken. Mere præcist er det i 'te Fibonacci-tal for $i \geq 0$ defineret ved

- $\text{Fib}(0) = 0$
- $\text{Fib}(1) = 1$
- $\text{Fib}(i) = \text{Fib}(i-2) + \text{Fib}(i-1)$, for $i \geq 2$.

Lav `Fib()` metode

- Lav en rekursiv `Fib()` metode ud fra ovenstående definition.
- Test din metode ved at checke, at $\text{Fib}(10) = 55$.
- Prøv din metode med forskellige inputs, eksempelvis 10, 20, 30, 40, 50, ...
 - Hvad sker der? Hvorfor?

Lav forbedret `Fib()` metode

- Set i lyset af erfaringerne gjort ovenfor, hvordan kunne `Fib()` så modificeres til at være usandsynligt meget mere effektiv?
- Implementér en sådan løsning og afprøv den med forskellige inputs, eksempelvis 10, 20, 30, 40, 50, ..., 100, 500, 1000, ...

Hvad er konklusionen på denne opgave? ☺

Øvelse 5.7: "Erathostenes' Si" (☆☆☆)

Denne øvelse går ud på at implementere en legendarisk metode til at finde alle primtal mindre end eller lig med et givet tal. Metoden går under navnet Erathostenes' Si, da den blev udtænkt ca. år 200 før kristus af den græske Erathostenes af Cyrene, der var både matematiker, digter, astronom og atlet. (dengang kunne de nogle ting! 😊)

Hans metode til at finde alle primtal mindre end eller lig med et givet heltal n udgøres af følgende skridt:

1. Lav en sorteret liste af tallene fra 2, 3, 4, ..., n
2. Initielt sættes $p = 2$ (som er det første primtal)
3. Streg i listen alle tal ud, som er multipla af p (og $\leq n$)
4. Find det første tilbageværende tal i listen, som er større end p
 - a. Dette tal – kald dette q – er det næste primtal
 - b. Sæt nu $p = q$
 - c. Hop tilbage til 3) + 4), hvis $p^2 \leq n$
 - i. Ellers gå videre til 5)
5. De tilbageværende tal i listen er algoritmens output: alle primtal $\leq n$

Implementér metoden

Opgaven er nu at implementere Erathostenes' algoritme i form af en C# metode `Erathostenes()`.

Module 6: "Introducing Object-Oriented Programming"

Øvelse 6.1: "BankAccount"

I denne øvelse vil vi lave vores første objekt-orienterede udvikling ved at fabrikere en klasse BankAccount, der modellerer en bank-konto, som vi kender den. Vi vil udstyre den med constructors, instans-metoder samt members.

Lav BankAccount-klassen

1. Lav et projekt i Visual Studio til ny C# konsol-applikation
 - a. Kald projektet "OOP" og
 - b. Placer det i
C:\Wincubate\87356\Module 6\Lab 6.1\Starter
2. Tilføj en ny fil "BankAccount.cs"
 - a. Højreklik OOP
 - b. Vælg **Add** -> **Class...**
 - c. Kald filen "BankAccount.cs"
 - d. Klik **Add**.
3. Lav BankAccount-klassen public og tilføj to member variable, der skal være private
 - a. `_accountNumber`, der er af type `int`
 - b. `_balance`, der er af type `decimal`.
4. Tilføj først en public constructor `BankAccount()`, der accepterer to argumenter og gemmer disse i de tilhørende member variabler
 - a. `accountNumber`, der er af type `int`
 - b. `initialBalance`, der er af type `decimal`.
 - i. Denne værdi skal være defaultet til 0, hvis den ikke angives på kaldtidspunktet
5. Når du er færdig, bør resultatet ligne

```
public class BankAccount
{
    private int _accountNumber;
    private decimal _balance;

    public BankAccount(int accountNumber, decimal initialBalance = 0)
    {
        _accountNumber = accountNumber;
        _balance = initialBalance;
    }
}
```

6. Byg programmet og korriger eventuelle fejl.

Tilføj metoder til BankAccount

7. Tilføj først en metode `GetBalance()`, der ikke tager argumenter, men returnerer en `decimal` som udgør kontoens saldo.
8. Tilføj dernæst en metode `Deposit()`, der accepterer ét argument
 - a. `amount`, der er af type `decimal`.I metode-kroppen skal `amount` lægges til kontoens saldo.

9. Tilføj dernæst en metode `Withdraw()`, der accepterer ét argument

- b. `amount`, der er af type `decimal`.

Metode skal returnere en værdi af type `bool`.

- Hvis der er nok penge på kontoen, fratrækkes `amount` fra saldoen, og der returneres `true`.
- Hvis ikke, foretages der ingen justeringer, og der returneres `false`.

10. Når du er færdig, bør resultatet ligne

```
public decimal GetBalance()
{
    return _balance;
}

public void Deposit(decimal amount)
{
    _balance += amount;
}

public bool Withdraw(decimal amount)
{
    if (_balance >= amount)
    {
        _balance -= amount;

        return true;
    }

    return false;
}
```

11. Byg programmet og korriger eventuelle fejl.

Afprøv BankAccount

12. I `Program.cs` skal `Main()` skrives, så

- a. Der oprettes en `BankAccount` med kontonummer 123456 med initielt 176 kr. på.
- b. Der indsættes 87 kr. på kontoen.
- c. Saldoen udskrives vha. `Console`.

13. Når du er færdig, bør resultatet ligne

```
static void Main(string[] args)
{
    BankAccount account = new BankAccount(123456, 176);
    account.Deposit(87);
    Console.WriteLine( account.GetBalance() );
}
```

14. Byg programmet og korriger eventuelle fejl.

15. Kør programmet og verificér, at 263 udskrives.

Øvelse 6.2: "Cross-language Objects" (★)

I denne opgave illustreres det, hvordan man fra C# kan kalde metoder på klasser, der er udviklet i andre .NET sprog som eksempelvis VB.NET.

Brug klasser fra C#

1. Åbn det eksisterende CrossLanguageObjects projekt i Visual Studio i
C:\Wincubate\87356\Module 6\Lab 6.2\Starter .
2. Verificér, at denne solution indeholder to forskellige projekter
 - a. Vb, der et class library skrevet i .NET-sproget VB.NET
 - b. Hello, der er en almindelig konsol-applikation skrevet i C#.
3. Verificér, at der i Vb i namespaces af samme navn indeholder en klasse Greetings med en metode Hello(), der returnerer en streng men ikke tager nogen parametre.
4. Byg din solution, så Vb bygges og kan refereres.
5. Der skal nu laves kode til at kalde denne metode fra Main() i Hello-projektet
 - a. Tilføj en reference fra Hello-projektet til Vb-projektet
 - i. Højre-klik **References** i Solution Explorer
 - ii. Vælg **Add Reference...**
 - iii. Vælg **Projects** tabben
 - iv. Dobbeltklik Vb i listen
 - b. Lokalisér "TODO 1: Invoke method on VB.NET class" i Hello-projektet
 - c. Lav en instans af Vb.Greetings klassen.
 - d. Kald Hello() på dette objekt og udskriv resultatet vha. Console.
6. Byg programmet og korriger eventuelle fejl.
7. Kør programmet og check at den korrekte hilsen fra VB.NET udskrives.

Undersøg integrationen af .NET-sprogene

8. Lokalisér "TODO 2: Modify" i Vb.
9. Prøv at ændre den returnerede værdi til noget andet.
10. Byg programmet og korriger eventuelle fejl.
11. Kør programmet og check, at den nye hilsen fra VB.NET udskrives.
12. Prøv til sidst at sætte et breakpoints i Main() i Hello i linien med kaldet til Vb
 - Verificér at man vha. debuggeren i Visual Studio kan steppe med ned i koden i projektet, selvom det er skrevet i et andet sprog.

Module 7: "Properties and Static Members"

Øvelse 7.1: "Properties"

I denne øvelse vil vi fortsætte BankAccount-eksemplet fra Øvelse 6.1, idet vil vi gøre indkapslingen bedre ved at benytte properties i stedet for metoder som tilgang til de forskellige member variabler.

Lav saldo property

1. Åbn det eksisterende OOP projekt i Visual Studio i
C:\Wincubate\87356\Module 7\Lab 7.1\Starter
som indeholder løsningen fra Øvelse 6.1.
2. Saldoen er en oplagt kandidat til en property.
3. Lav `_balance` til en property af type `decimal`, der hedder `Balance`
 - a. `get` skal være `public`
 - b. `set` skal være `private`
4. Metoden `GetBalance()` er nu overflødig og skal fjernes.
5. Justér resten af `BankAccount` samt `Main()`-metoden, således at `Balance` alle steder benyttes som property i stedet for member variabel og metode-kald. (NB: Dog ikke i kroppen af propertyens `get` og `set`).
6. Når du er færdig, bør resultatet ligne

```
static void Main(string[] args)
{
    BankAccount account = new BankAccount(123456, 176);
    account.Deposit(87);
    Console.WriteLine( account.Balance );
}
```

```
public class BankAccount
{
    public BankAccount(int accountNumber, decimal initialBalance = 0)
    {
        _accountNumber = accountNumber;
        Balance = initialBalance;
    }

    public void Deposit(decimal amount)
    {
        Balance += amount;
    }
}
```

```

public bool Withdraw(decimal amount)
{
    if (Balance >= amount)
    {
        Balance -= amount;

        return true;
    }
    return false;
}

private int _accountNumber;

public decimal Balance
{
    get
    {
        return _balance;
    }
    private set
    {
        _balance = value;
    }
}

private decimal _balance;
}

```

7. Byg programmet og korriger eventuelle fejl.
8. Kør programmet og verificér, at 263 stadig korrekt udskrives.

Lav kontonummer property

På samme vis som ovenfor bør `_accountNumber` laves til en property. Du kan få god hjælp af Visual Studio.

9. Lav `BankAccount._accountNumber` til en property som beskrevet ovenfor
 - a. Højreklik `_accountNumber`, hvor den er erklæret
 - b. Vælg **Quick Actions...**
 - c. Vælg **Encapsulate Field**
 - d. Klik **OK**.
10. Omrokér koden lidt, hvis du synes, at det er pænere. Justér synligheden af propertyen
 - a. `get` skal være `public`
 - b. `set` skal være `private`
11. Når du er færdig, bør resultatet ligne


```

public BankAccount(int accountNumber, decimal initialBalance = 0)
{
    AccountNumber = accountNumber;
    Balance = initialBalance;
}

...

public int AccountNumber
{
    get
    {
        return _accountNumber;
    }
    private set
    {
        _accountNumber = value;
    }
}
private int _accountNumber;

```

12. Byg programmet og korriger eventuelle fejl.
13. Kør programmet og verificér, at 263 stadig korrekt udskrives.

Lav om til automatiske properties

De to properties kunne have været lavet som automatiske properties.

14. Lav til sidst AccountNumber og Balance om til automatiske properties.
15. Når du er færdig, bør resultatet ligne

```

public int AccountNumber { get; private set; }
public decimal Balance { get; private set; }

```

16. Byg programmet og korriger eventuelle fejl.

Kør programmet og verificér for allersidste gang, at 263 stadig korrekt udskrives.

Øvelse 7.2: "Statiske klasser, metoder og data" (★)

Denne øvelse er en fortsættelse af BankAccount-klassen fra Øvelse 7.1. Vi vil her arbejde med statiske aspekter af klasser. Først lave statiske data og metode på BankAccount, hvorefter der laves en statisk hjælpeklasse Bank.

Lav static counter og metode for næste kontonummer

1. Åbn det eksisterende OOP projekt i Visual Studio i
C:\Wincubate\87356\Module 7\Lab 7.2\Starter
som indeholder løsningen fra Øvelse 7.1.
2. Tilføj til BankAccount en private static variabel af typen int og kald den s_NextAccountNumber.
3. Lav en private static metode GetNextAccountNumber(), der ikke accepterer argumenter, men returnerer s_NextAccountNumber og tæller den op.
4. Fjern int accountNumber argumentet fra constructoren i BankAccount, da kontonummeret nu skal automatisk beregnes.
5. Sørg for, at disse constructors nu benytter GetNextAccountNumber() til at sætte AccountNumber på nyskabte instanser af BankAccount.
6. Når du er færdig, bør resultatet ligne

```
private static int s_NextAccountNumber;

private static int GetNextAccountNumber()
{
    return s_NextAccountNumber++;
}

public BankAccount(decimal initialBalance = 0)
{
    AccountNumber = GetNextAccountNumber();
    Balance = initialBalance;
}
```

7. Ret Main() til at reflektere disse ændringer.
8. Udvid koden i Main() til at udskrive account.AccountNumber.
9. Når du er færdig, bør resultatet ligne

```
static void Main(string[] args)
{
    BankAccount account = new BankAccount(176);
    Console.WriteLine(account.AccountNumber);
    account.Deposit(87);
    Console.WriteLine( account.Balance );
}
```

10. Byg programmet og korriger eventuelle fejl.
11. Kør programmet
 - a. Verificér, at 263 stadig korrekt udskrives.
 - b. Hvad sker der med kontonummeret?

Lav static constructor for BankAccount

10. Lav en static constructor i BankAccount, der sørger for, at kontonumrene starter fra 100000.
11. Når du er færdig, bør resultatet ligne

```
static BankAccount()  
{  
    s_NextAccountNumber = 100000;  
}
```

12. Byg programmet og korriger eventuelle fejl.
13. Kør programmet og verificér, at kontonummeret er korrekt.

Lav static hjælpeklasse Bank

14. Tilføj på samme måde som i Øvelse 6.1 en ny klasse Bank ved at tilføje en ny fil Bank.cs.
15. Lav Bank static.
16. Tilføj til Bank en public static metode CreateAccount(), der accepterer ét argument initialAmount af type decimal, og returnerer et nyskabt BankAccount objekt med initialAmount som saldo.
17. Tilføj til Bank en public static metode TransferFunds(), der accepterer tre argumenter
 - a. from af type BankAccount
 - b. amount af type decimal
 - c. to af type BankAccount

Metoden skal overføre amount fra from-kontoen til to-kontoen. Hvis dette er muligt, returneres true, og false ellers.

18. Når du er færdig, bør resultatet ligne

```
static class Bank  
{  
    public static BankAccount CreateAccount( decimal initialAmount )  
    {  
        return new BankAccount(initialAmount);  
    }  
  
    public static bool TransferFunds( BankAccount from, decimal amount,  
                                     BankAccount to )  
    {  
        if (from.Withdraw(amount))  
        {  
            to.Deposit(amount);  
  
            return true;  
        }  
  
        return false;  
    }  
}
```

19. Byg programmet og korriger eventuelle fejl.
20. Omskriv Main() til at afprøve Bank og dens metoder.
21. Når du er færdig, kunne resultatet eventuelt ligne

```

static void Main(string[] args)
{
    BankAccount account = Bank.CreateAccount(176);
    BankAccount otherAccount = Bank.CreateAccount(1024);

    if (Bank.TransferFunds(otherAccount, 555, account))
    {
        Console.WriteLine("Transferred funds from {0} to {1}",
            otherAccount.AccountNumber, account.AccountNumber);

        Console.WriteLine("{0} now contains {1}",
            account.AccountNumber, account.Balance);
        Console.WriteLine("{0} now contains {1}",
            otherAccount.AccountNumber, otherAccount.Balance);
    }
}
}

```

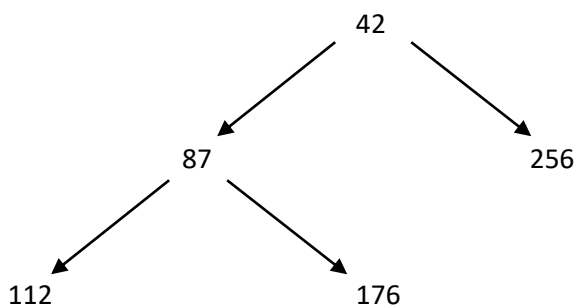
22. Byg programmet og korriger eventuelle fejl.
23. Kør programmet og se veltilfreds på, at alt fungerer perfekt... 😊

Notér: Man burde have en hjælpemetode til at udskrive indholdet af et `BankAccount` objekt. Én mulighed er at lave endnu en hjælpemetode i `Bank`. Dette er dog ikke den helt rigtige løsning, som vi skal se i Module 8...

Øvelse 7.3: "Rekursive datatyper" (☆☆)

I denne opgave vil vi beskæftige os med rekursive datatyper. Vi vil lave en klasse `Node`, der selv kan referere instanser af `Node`. **Bemærk** at `structs` i `C#` ikke er tilladt at være rekursive!

`Node` skal udgøre knuderne i et binært træ, som for eksempel



Hver af knuderne i træet har en heltalsværdi, der kan repræsenteres som en `int`.

Lav `Node()` klassen

- Definér klassen `Node`.
- Lav to constructors
 - Én der accepterer et heltal, der er knudens værdi
 - Én der accepterer et heltal samt to instanser af `Node`
 - Brug constructor chaining!

Lav binært træ

- Lav en variabel `node` af type `Node`.
- Skriv et udtryk, der sætter `node` til at indeholde det binære træ afbildet ovenfor.

Udskriv træet

- Lav en metode `Print()` på `Node`-klassen, der gennemløber træet og udskriver værdierne i knuderne.
- Test din metode på `node` defineret ovenfor
 - Den udskrevne sekvens af værdier for `node.Print()` skal være 42, 87, 112, 176, 256.

Alternativ udskrift

Gennemløbet af træet, som du lavede ovenfor, kaldes et *prefix-gennemløb*.

- Overvej hvordan du kan lave en alternativt `Print()` metode, så den udskrevne sekvens bliver 112, 176, 87, 256, 42.

Module 8: “Inheritance and Polymorphism”

Øvelse 8.1: “Nedarvning og klasse-hierarkier”

I denne opgave vil vi beskæftige os med nedarvning og hierarkier af klasser, der modellerer geometriske begreber. Vi vil forsyne disse med virtuelle properties og metoder og overskrive disse passende ned gennem hierarkiet.

Lav Shape klasse

1. Lav et projekt i Visual Studio til ny C# konsol-applikation
 - a. Kald projektet ”Shapes” og
 - b. Placér det i
C:\Wincubate\87356\Module 8\Lab 8.1\Starter
2. Tilføj en ny fil ”Shape.cs”.
3. Lav her en abstract klasse Shape
 - c. Shape skal indeholde en enkelt abstract og public property Area af type double med en get.
4. Når du er færdig, bør resultatet ligne

```
abstract class Shape
{
    abstract public double Area { get; }
}
```

5. Byg programmet og korriger eventuelle fejl.

Lav Circle klasse

6. Tilføj en ny fil ”Circle.cs”.
7. Lav her en klasse Circle, der arver fra Shape
 - a. Circle skal have en automatisk property Radius af typen double med en public get og en protected set.
 - b. Circle skal have en constructor, der accepterer et argument radius af typen double og sætter den tilsvarende property.
 - c. Circle skal desuden overskrive Area’s get til at returnere det beregnede areal af cirklen
 - i. Tip: En cirkels areal er πr^2 , hvor r er radius. Benyt Math-klassen til at få en værdi for π .
8. Når du er færdig, bør resultatet ligne

```

class Circle : Shape
{
    public double Radius { get; protected set; }

    public Circle(double radius)
    {
        Radius = radius;
    }

    public override double Area
    {
        get
        {
            return Math.PI * Radius * Radius;
        }
    }
}

```

9. Byg programmet og korriger eventuelle fejl.

Tilføj en Rectangle

10. Tilføj på samme måde en Rectangle-klasse, der arver fra Shape
- a. Rectangle skal have en automatisk property Width af typen double med en public get og en protected set.
 - a. Tilføj en lignende Height property.
 - b. Lav en constructor, der accepterer to argumenter width og height, og sætter de tilsvarende properties
 - c. Overskriv Area's get passende
11. Når du er færdig, bør resultatet ligne

```

class Rectangle : Shape
{
    public double Width { get; protected set; }
    public double Height { get; protected set; }

    public Rectangle( double width, double height )
    {
        Width = width;
        Height = height;
    }

    public override double Area
    {
        get
        {
            return Width * Height;
        }
    }
}

```


Beregn arealer for forskellige Shapes

12. I Program.cs skal Main() skrives, så
 - a. Der oprettes et array af Shape objekter kaldet all indeholdende
 - i. En Circle med radius 3
 - ii. En Rectangle med siderne 4 og 5
 - b. Der skal itereres gennem all og for hver Shape udskrives følgende
Arealet af *shape* er *areal*
13. Når du er færdig, bør resultatet ligne

```
static void Main(string[] args)
{
    Shape[] all = { new Circle(3), new Rectangle(4, 5) };

    foreach (Shape s in all)
    {
        Console.WriteLine( "Arealet af {0} er {1}",
            s,
            s.Area );
    }
}
```

14. Byg programmet og korriger eventuelle fejl.
15. Kør programmet og verificér, at programmet udskriver de korrekte værdier for areal
 - c. Notér dig, at dette er polymorfi i fuld flor! ☺
16. Hvad skrives der for *shape*? Hvorfor...?

Overskriv ToString() for de forskellige Shapes

17. Overskriv ToString() for Circle til at skrive
Cirkel med Radius *radius*
18. Overskriv ToString() for Rectangle til at skrive
Rektangel med Sidelængder *width* og *height*
19. Når du er færdig, bør resultatet ligne

```
class Circle : Shape
{
    ...

    public override string ToString()
    {
        return "Cirkel med Radius " + Radius;
    }

    ...
}
```

```

class Rectangle : Shape
{
    ...

    public override string ToString()
    {
        return string.Format("Rektangel med Sidelængder {0} og {1}",
            Width,
            Height );
    }

    ...
}

```

20. Byg programmet og korriger eventuelle fejl.
21. Kør programmet og verificér, at programmet nu udskriver de korrekte værdier for både objekt og areal.

Lav Square klasse

22. Tilføj en ny fil "Square.cs".
23. Lav her en klasse Square, der arver fra Rectangle
 - a. Square skal have en constructor, der accepterer et argument width af typen double og kalder Rectangle's constructor med width som begge side-størrelser.
 - b. Bemærk at Area så allerede er korrekt..! Intet mindre end fantastisk! ☺
 - c. Overskriv ToString() for Square til at skrive
Kvadrat med Sidelængde *width*
24. Når du er færdig, bør resultatet ligne

```

class Square : Rectangle
{
    public Square( double width ) : base( width, width )
    {
    }

    public override string ToString()
    {
        return string.Format("Kvadrat med Sidelængde {0}",
            Width );
    }
}

```

25. Tilføj til all i Main() to Square-objekter med sider 6 og 7, hhv.
26. Når du er færdig, bør resultatet ligne

```

Shape[] all = { new Circle(3), new Rectangle(4,5), new Square(6),
                new Square(7) };

```

27. Byg programmet og korriger eventuelle fejl.
28. Kør programmet og verificér, at programmet nu udskriver de korrekte værdier for både objekt og areal.

- d. Læg mærke til, at `Main()` udover ovenstående tilføjelse slet ikke skulle ændres..!
29. Byg programmet og korriger eventuelle fejl.

Hvis du har masser af energi...

...så kan du selv udvide applikationen med flere geometriske figurer.

Prøv for eksempel at kigge på listen på <http://www.mathsisfun.com/area.html> og slå dig løs! ☺

Øvelse 8.2 "Nedarvning af klasser i andre sprog" (★)

Denne øvelse illustrerer hvorledes man kan nedarve klasser fra andre .NET-sprog og overskrive virtuelle metoder.

Dette er en fortsættelse af Øvelse 6.2, hvor vi lavede objekter af klasser skrevet i VB.NET og kaldte metoder på dem. Hvis du ikke fik prøvet kræfter med denne opgave under øvelserne til Module 6, så kig kort pointerne i Øvelse 6.2 igennem først. Hvis muligt, løs hele Øvelse 6.2...

Indledning

1. Åbn det eksisterende CrossLanguageObjects projekt i Visual Studio i
C:\Wincubate\87356\Module 8\Lab 8.2\Starter .
2. Verificér, at denne solution indeholder to forskellige projekter
 - a. Vb, der et class library skrevet i .NET-sproget VB.NET
 - b. Hello, der er en almindelig konsol-applikation skrevet i C#.
3. Kør programmet og check, at den korrekte hilsen fra VB.NET udskrives.

Nedarvning og overskrivning af Hello()

4. Vb.Greetings skal nu nedarves og Hello() metoden skal overskrives
 - a. Lokalisér "TODO 1: Inherit from Vb.Greetings" i Greetings.cs.
 - b. Tilføj en C# klasse Greetings, der nedarver VB.NET-klassen.
 - c. Overskriv Hello() metoden i Greetings, således at den returnerer den oprindelige hilsen fra Vb.Greetings påklippet et tidsstempel fra kaldtidspunktet.
 - i. DateTime.Now.ToLongTimeString() udgør et sådant tidsstempel
5. Når du er færdig, bør resultatet ligne

```
// TODO 1: Inherit from Vb.Greetings
class Greetings : Vb.Greetings
{
    public override string Hello()
    {
        return base.Hello() + " " + DateTime.Now.ToLongTimeString();
    }
}
```

6. Byg programmet og korriger eventuelle fejl.
7. Herefter skal der laves en instans af den nye C# Greetings-klasse og Hello() skal kaldes.
 - d. Fjern de to linier med det eksisterende kald til Hello() på Vb.Greetings.
 - e. Lav en ny instans af Greetings i en lokal variabel csGreetings.
 - f. Udskriv resultatet af csGreetings.Hello().
8. Når du er færdig, bør resultatet ligne

```
static void Main(string[] args)
{
    // TODO 2: Invoke overridden Hello() method
    Greetings csGreetings = new Greetings();
    Console.WriteLine( csGreetings.Hello() );
}
```

9. Byg programmet og korriger eventuelle fejl.
10. Kør programmet og verificér, at den oprindelige hilsen fra VB.NET udskrives efterfulgt af det aktuelle tidspunkt.

Lidt eksperimenter...

11. Prøv at ændre den returnerede værdi i Vb til noget andet.
12. Kør programmet og verificér, at den modificerede hilsen fra VB.NET udskrives efterfulgt af det aktuelle tidspunkt.

Øvelse 8.3 "Pæn udskrivning af BankAccount" (☆☆)

Hvis du synes, at det gjorde ondt i hjertet i Øvelse 7.2 at skrive ting som

```
Console.WriteLine("{0} now contains {1}",  
    account.AccountNumber, account.Balance);  
Console.WriteLine("{0} now contains {1}",  
    otherAccount.AccountNumber, otherAccount.Balance);
```

ja, så havde du helt ret. Man bør ikke gentage sådan formateringskode igen og igen.

- Åbn det eksisterende OOP projekt i Visual Studio i
C:\Wincubate\87356\Module 8\Lab 8.3\Starter
hvilket indeholder løsningen til Øvelse 7.2.
- Overskriv `BankAccount.ToString()` passende, så du igen kan sove trygt om natten...
- Modificér `Main()` til at udskrive objekterne direkte.

Øvelse 8.4 "Virtual vs. new" (☆☆☆)

Betragt følgende klasse-definitioner:

```
class A
{
    public virtual void M()
    {
        Console.WriteLine("A");
    }
}

class B : A
{
    public override void M()
    {
        Console.WriteLine("B");
    }
}

class C : B
{
    new public virtual void M()
    {
        Console.WriteLine("C");
    }
}

class D : C
{
    public override void M()
    {
        Console.WriteLine("D");
    }
}
```

```
class Program
{
    public static void Main()
    {
        D d = new D();
        C c = d;
        B b = c;
        A a = b;

        d.M();
        c.M();
        b.M();
        a.M();
    }
}
```

- Tænk over – uden at køre programmet!! – hvad ovenstående program mon udskriver ?
 - Hvorfor?
- Samme spørgsmål i det tilfælde, hvor alle fire M() metoder var erklæret virtual?
- Samme spørgsmål i det tilfælde, hvor alle fire M() metoder var erklæret new...?

Øvelse 9.1: "Exception-håndtering i Bank"

Indtast beløb

1. Åbn det eksisterende OOP projekt i Visual Studio i
C:\Wincubate\87356\Module 9\Lab 9.1\Starter .
2. Kør programmet og se, hvad der foregår.
3. Beløbet i amount skal nu indtastes via Console
 - g. Lokalisér "TODO 1: Input amount to transfer"
 - h. Erklær en variabel amount af type decimal og sæt den til at indeholde den indlæste værdi
 - i. Sørg for, at amount er det beløb, som skal overflyttes mellem de to konti.
4. Når du er færdig, bør resultatet ligne

```
// TODO 1: Input amount to transfer
Console.WriteLine("Input amount to be transferred: ");
decimal amount = decimal.Parse(Console.ReadLine());

if (Bank.TransferFunds(otherAccount, amount, account))
{
    Console.WriteLine("Transferred funds from {0} to {1}",
        otherAccount, account);
}
```

- ## Lav exception-håndtering

- 65

9. Når du er færdig, bør resultatet ligne

```
try
{
    Console.WriteLine("Input amount to be transferred: ");
    decimal amount = decimal.Parse(Console.ReadLine());

    if (Bank.TransferFunds(otherAccount, amount, account))
    {
        Console.WriteLine("Transferred funds from {0} to {1}",
            otherAccount, account);
    }
}
catch
{
    Console.WriteLine( "Something went wrong..." );
}
```

10. Byg programmet og korriger eventuelle fejl.
11. Hvad sker der nu, når programmet kører med ovenstående tre inputs?
12. For eksemplets skyld vil vi undersøge, hvilke informationer, der findes på Exception-klassen.
- Modificér catch-blokken til – for alle exceptions – at udskrive
Exception.Message
Exception.Source
Exception.StackTrace

13. Når du er færdig, bør resultatet ligne

```
try
{
    ...
}
catch (Exception exception)
{
    string error = string.Format(
        "An error occurred processing amount." + Environment.NewLine +
        "Message:    {0}" + Environment.NewLine +
        "Source:      {1}" + Environment.NewLine +
        "StackTrace: {2}" + Environment.NewLine,
        exception.Message,
        exception.Source,
        exception.StackTrace );
    Console.WriteLine( error );
}
```

14. Byg programmet og korriger eventuelle fejl.
15. Hvad sker der nu, når programmet kører med ovenstående tre inputs?
16. Vi kunne dog godt tænke os en mere detaljeret fejlmeddelelse, så vi vil fange hver formateringsfejl særskilt
- Fjern den nyligt tilføjede catch(Exception exception) blok, da denne mest af alt indeholder informationer for programmøren og ikke brugeren!
 - Tilføj en catch-blok, der fanger FormatException med en passende beskrivelse

17. Når du er færdig, bør resultatet ligne

```
try
{
    ...
}
catch( FormatException )
{
    Console.WriteLine( "Input amount is not in correct format" );
}
```

18. Byg programmet og korriger eventuelle fejl.

19. Hvad sker der nu, når programmet kører med ovenstående tre inputs?

20. Vi vil nu fange også overløbsfejl særskilt

d. Tilføj en catch-blok, der fanger OverflowException med en passende beskrivelse

21. Når du er færdig, bør resultatet ligne

```
try
{
    ...
}
catch( FormatException )
{
    Console.WriteLine( "Input amount is not in correct format" );
}
catch (OverflowException)
{
    Console.WriteLine("Input amount is an exceedingly large-sized amount");
}
```

22. Byg programmet og korriger eventuelle fejl.

23. Hvad sker der nu, når programmet kører med ovenstående tre inputs?

Tilføjelse af finally-blok

Ofte vil det være fordelagtigt at få udført noget kode, inden processeringen stopper – uanset om der kastes exceptions eller ej!

24. Tilføj en finally-blok, der udskriver, at processeringen er stoppet.

25. Når du er færdig, bør resultatet ligne

```
try
{
    ...
}
catch( FormatException )
{
    Console.WriteLine( "Input amount is not in correct format" );
}
catch (OverflowException)
{
    Console.WriteLine("Input amount is an exceedingly large-sized amount");
}
finally
{
    Console.WriteLine("Ending processing of amount");
}
```

26. Byg programmet og korriger eventuelle fejl.

27. Hvad sker der nu, når programmet kører med ovenstående tre inputs mht. finally?

Yderligere exception-håndtering

Man bør dog også håndtere selv de mest uforudsete situationer. For at afprøve dette i praksis, gøres følgende:

28. Erstat nu kodelinien (eller kodelinierne), der indlæser og parser amount til at være nedenstående

```
string input = Console.ReadLine();
decimal amount = decimal.Parse( input );
```

29. Sæt et breakpoint i den nederste af de to linier

30. Byg programmet og korriger eventuelle fejl

31. Kør programmet. Når breakpointet mødes, ændr da værdien af input til at være null.

32. Fortsæt programmet

- a. Hvad sker der nu?
- b. Hvad med finally?

33. Håndtér også denne type fejl ved at tilføje en passende catch-blok.

34. Når du er færdig, bør resultatet ligne

```
try
{
    ...
}
catch( FormatException )
{
    Console.WriteLine( "Input amount is not in correct format" );
}
catch( OverflowException )
{
    Console.WriteLine("Input amount is an exceedingly large-sized amount");
}
catch
{
    Console.WriteLine( "An internal error has happened" );
}
finally
{
    Console.WriteLine("Ending processing of amount");
}
```

35. Kør programmet igen som du gjorde tidligere (med breakpoint samt null-værdi osv.).

36. Hvad sker der så nu?

Øvelse 9.2 "TryParse" (★)

Den allerbedste håndtering af exceptions er dog så vidt muligt at undgå dem!

Det er faktisk sådan, at der på `decimal` (og alle de andre simple værdityper) findes en `TryParse()`-metode, som ikke kaster de exceptions, som vi i Øvelse 9.1 brugte mange kræfter at håndtere.

Indledning

1. Åbn det eksisterende OOP projekt i Visual Studio i
 `C:\Wincubate\87356\Module 9\Lab 9.2\Starter` ,
 der tager udgangspunkt i løsningen for Øvelse 9.1.
2. Projektet skal nu benytte `TryParse()` i stedet
 - a. Lokalisér "TODO 1: Use decimal.TryParse() instead"
 - b. Undersøg vha. de indbyggede hjælpe-faciliteter hvordan `decimal.TryParse()` virker
 - c. Omskriv `Main()` til at benytte denne metode på "passende" vis.
3. Byg programmet og korriger eventuelle fejl.
4. Kør programmet og prøv med samtlige inputs og teknikker fra Øvelse 9.1.
5. Hvilke exception-håndteringer er nu nødvendige?

Øvelse 9.3 "Egne Exception-klasser" (★)

Ofte er de indbyggede exceptions ikke betegnende for den fejl, som man gerne vil informere om. Derfor får man jævnligt brug for at definere sine egne exception-klasser. Disse bør så kastes og fanges i egne klasse i stedet for at returnere en boolean, der angiver om operationen gik godt.

Vi fortsætter her fra Øvelse 9.1 og 9.2 ved at definere en klasse `InsufficientFundsException`, der arver fra `Exception`.

Definér Exception-klasse

1. Åbn det eksisterende OOP projekt i Visual Studio i
C:\Wincubate\87356\Module 9\Lab 9.3\Starter ,
der tager udgangspunkt i løsningen for Øvelse 9.2.
2. Tilføj en ny fil "`InsufficientFundsException.cs`".
3. Lav her en klasse `InsufficientFundsException`, der arver fra `Exception`
 - a. `InsufficientFundsException` skal have en public constructor uden parametre
 - b. `InsufficientFundsException` skal have en public constructor, der tager en string message som parameter og kalder `Exception`'s tilsvarende constructor
 - c. `InsufficientFundsException` skal have en public constructor, der tager en string message og `Exception` inner som parametre og kalder `Exception`'s tilsvarende constructor.
4. Lav i klasse `InsufficientFundsException` en private readonly member variabel af typen `BankAccount`.
5. Lav en read-only property af typen `BankAccount`, der hedder `Account` og returnerer den underliggende member variabel.
6. En fjerde constructor skal så tilføjes
 - a. `InsufficientFundsException` skal have en public constructor, der tager en string message og en `BankAccount account` som parametre, sætter member variablen og kalder den tilsvarende constructor på message.
7. Når du er færdig, bør resultatet ligne

```

class InsufficientFundsException : Exception
{
    public BankAccount Account
    {
        get { return _account; }
    }
    private readonly BankAccount _account;

    public InsufficientFundsException()
    {
    }

    public InsufficientFundsException( string message ) : base( message )
    {
    }

    public InsufficientFundsException( string message, Exception inner )
        : base( message, inner )
    {
    }

    public InsufficientFundsException( string message,
        BankAccount account) : this( message )
    {
        _account = account;
    }
}

```

8. Byg programmet og korriger eventuelle fejl.

Modificér BankAccount

9. Modificér nu BankAccount.Withdraw() til at kaste en InsufficientFundsException med konto-referencen, når operationen ikke kan gennemføres.
10. Når du er færdig, bør resultatet ligne

```

class BankAccount
{
    public void Withdraw(decimal amount)
    {
        if (Balance >= amount)
        {
            Balance -= amount;
        }
        else
        {
            throw new InsufficientFundsException("Could not complete
withdrawal from account", this);
        }
    }
}

```


11. Byg programmet og korriger eventuelle fejl.

Modificér Bank

12. Modificér nu ligeledes `Bank.TransferFunds()` til at kaste en `InsufficientFundsException` med konto-referencen, når operationen ikke kan gennemføres.
13. Når du er færdig, bør resultatet ligne

```
static class Bank
{
    ...

    public static void TransferFunds( BankAccount from, decimal amount,
                                     BankAccount to )
    {
        from.Withdraw(amount);
        to.Deposit(amount);
    }

    ...
}
```

14. Byg programmet og korriger eventuelle fejl.

Modificér Main()

15. Modificér nu `Main()` til at fange en `InsufficientFundsException` og håndtere situationen passende.
16. Når du er færdig, bør resultatet ligne

```

static void Main(string[] args)
{
    BankAccount account = Bank.CreateAccount(176);
    BankAccount otherAccount = Bank.CreateAccount(1024);

    try
    {
        Console.WriteLine("Input amount to be transferred: ");
        string input = Console.ReadLine();

        decimal amount;
        if (decimal.TryParse(input, out amount))
        {
            Bank.TransferFunds(otherAccount, amount, account);

            Console.WriteLine("Transferred funds from {0} to {1}",
                otherAccount, account);
        }
        else
        {
            Console.WriteLine("Input amount can be recognized as a legal
number to transfer");
        }
    }
    catch (InsufficientFundsException exception)
    {
        Console.WriteLine( "Error processing {0}: {1}",
            exception.Account,
            exception.Message );
    }
    catch
    {
        Console.WriteLine("An internal error has happened");
    }
    finally
    {
        Console.WriteLine("Ending processing of amount");
    }
}

```

17. Byg programmet og korriger eventuelle fejl.

18. Kør programmet med

- a. 555
- b. 555555

som input og observer, at alt foregår, som du forventer det...

Øvelse 9.4 "BankException" (☆☆)

Man kan overveje, at lave et endnu mere indpakket exception-design ved at introducere en `BankException`, som relaterer sig til de operationer, som ligger i `Bank`-klassen.

Denne øvelse, som er en fortsættelse af Øvelse 9.3, gør netop dette.

- Åbn det eksisterende OOP projekt i Visual Studio i
 `C:\Wincubate\87356\Module 9\Lab 9.4\Starter`
 hvilket indeholder løsningen til Øvelse 9.3.
- Lav en `BankException`.
- Modificér `Bank.TransferFunds()` til at
 - Fange `InsufficientFundsException`
 - "Ompakke" den til en ny `BankException`, der kastes med den originale exception som indre exception.
- Modificér `Main()` passende.
 - Tilgå evt. `BankException.InnerException`.

Øvelse 9.5 "To Throw or Not To Throw" (☆☆☆)

Hvad er mon forskellen på disse to kodestumper?

```
try
{
    ...
}
catch (InsufficientFundsException exception)
{
    throw exception;
}
```

```
try
{
    ...
}
catch (InsufficientFundsException)
{
    throw;
}
```

Module 10: "Interfaces"

Øvelse 10.1: "IEnumerable på Deck-typen og IComparable på Card-typen"

Vi vil nu gå skridtet videre og indpakke de 52 kort af den velkendte Card-type i en ny klasse, Deck, der indeholder et array af 52 kort. Vi vil så implementere IEnumerable på Deck og se, hvorledes vi derefter med foreach kan iterere gennem alle kortene i spillet. Derefter vil vi implementere IComparable på Card, så de forskellige kort kan sorteres.

Inspicér Deck

1. Åbn det eksisterende DeckTest projekt i Visual Studio i
C:\Wincubate\87356\Module 10\Lab 10.1\Starter
hvilket indeholder en omstrukturering af allerede kode, som vi tidligere har implementeret.
2. Inspicér først Card.cs og se, at den indeholder en struct Card, som vi kender og elsker den.
 - a. Læg mærke til, hvordan ToString() nu er overskrevet til at benytte udskrivningen, som vi lavede i øvelserne til Module 4 og 5.
3. Inspicér dernæst Deck.cs og se, at den indeholder en helt ny klasse Deck, som laver indpakningen af 52 elementer af type Card i en separat klasse.
 - a. Bemærk at der findes en ny constructor, der initialiserer Deck til at indeholde et helt spil kort vha. samme metoder, som vi skrev i øvelserne til Module 4 og 5, dvs. den tidligere statiske FillDeck()
 - b. Læg desuden mærke til metoden Shuffle(), der blander kortspillet præcis som ShuffleDeck() gjorde det i øvelserne til Module 4 og 5.
4. Byg programmet, og check at det kompilerer.

Implementér IEnumerable

5. Åbn filen Deck.cs og Lokalisér "TODO 1: Implement IEnumerable" i koden
 - a. tilføj til definitionen, at Deck skal implementere IEnumerable-interfaceet
 - i. Skriv " : IEnumerable" og lad eventuelt Visual Studio hjælpe dig. 😊
6. Herefter skal metode-kroppen til GetEnumerator() udfyldes
 - b. Returnér inde i metoden den IEnumerator, som allerede fra .NETs hånd findes i arrayet i _cards.
7. Når du er færdig, bør resultatet ligne

```
public IEnumerator GetEnumerator()  
{  
    return _cards.GetEnumerator();  
}
```

8. Byg programmet og korriger eventuelle fejl.
9. I Program.cs skal Main() opdateres, så alle kort i Deck-instansen udskrives med en foreach-sætning
 - c. Find "TODO 2: Iterate through all cards with foreach" i koden og lav implementationen
 - d. Bemærk, at Card nu kan udskrives direkte via Console.WriteLine().
10. Når du er færdig, bør resultatet ligne

```
Deck deck = new Deck();
deck.Shuffle();

// TODO 2: Iterate through all cards with foreach
foreach ( Card card in deck )
{
    Console.WriteLine( card );
}
```

11. Byg programmet og korriger eventuelle fejl.
12. Kør programmet og verificér, at alle kort i et blandet sæt kort udskrives.

Implementér IComparable

13. Åbn filen Deck.cs og Lokalisér "TODO 3: Implement method" i koden
 - a. Metode-kroppen af Arrange()-metoden skal nu implementeres, så det underliggende array af kort sorteres med Array.Sort().

14. Når du er færdig, bør resultatet ligne

```
public void Arrange()
{
    Array.Sort( _cards );
}
```

15. I Program.cs skal Main() igen opdateres, så Arrange() kaldes, hvorefter Deck-instansen igen udskrives med en foreach-sætning.
 - b. Lokalisér "TODO 4: Arrange and display Deck" i koden og lav implementationen.
16. Når du er færdig, bør resultatet ligne

```
deck.Arrange();

Console.WriteLine("Rearranged Deck: ");
foreach ( Card card in deck )
{
    Console.WriteLine( card );
}
```

17. Byg programmet og korriger eventuelle fejl.
18. Kør programmet og observer resultatet
 - c. Hvad sker der, når du kører programmet?
 - d. Hvorfor...?
19. Åbn filen Card.cs og lokalisér "TODO 5: Implement IComparable" i koden
 - e. tilføj til definitionen, at Card skal implementere IComparable-interfacet
 - i. Benyt Visual Studio til at implementere interfacet på samme måde som med IEnumerable.

Der laves nu tomme skaller til metoden i IComparable.
20. Herefter skal metode-kroppen til CompareTo() udfyldes, således at kortene i sorteret tilstand vil forekomme som i et ny-initialiseret Deck-objekt
 - f. Check først, at obj er af typen Card.

- i. Hvis ikke skal der kastes en `ArgumentException` med teksten
Can only compare Card to other Cards
 - ii. Bemærk at `as`-keywordet ikke kan bruges her, da `Card` er en værdi-type!
 - iii. Benyt i stedet `is`-keywordet med et efterfølgende `cast` til at konvertere `obj` til typen `Card`
 - g. Sammenlign først `this.suit` med `other.suit`
 - i. Hvis `this.suit` er mindre, returneres -1
 - ii. Hvis `this.suit` er større, returneres 1
 - iii. Hvis de er ens, skal `this.rank` sammenlignes med `other.rank`
 - 1. Hvis `this.rank` er mindre, returneres -1
 - 2. Hvis `this.rank` er større, returneres 1
 - 3. Ellers returneres 0.
21. Når du er færdig, bør resultatet ligne

```
#region IComparable Members

public int CompareTo(object obj)
{
    if (obj is Card)
    {
        Card other = (Card) obj;
        if (this.suit < other.suit) { return -1; }
        else if (this.suit > other.suit) { return 1; }
        else
        {
            if (this.rank < other.rank) { return -1; }
            else if (this.rank > other.rank) { return 1; }
            else { return 0; }
        }
    }

    throw new ArgumentException(
        "Can only compare Card to other Cards");
}

#endregion
```

- 22. Byg programmet og korriger eventuelle fejl.
- 23. Kør programmet endnu en gang og observér resultatet
 - h. Hvad sker der nu, når du kører programmet?
 - i. Hvorfor...?

Øvelse 10.2: "Hold øje med garbage collection"

I denne øvelse vil vi prøve at få en fornemmelse af, hvornår garbage collectoren sætter ind for at rydde objekter op. Vi vil definere en klasse A, der kan tælle antallet af sine instanser. Dette antal vil vi så løbende skrive ud, så vi kan følge lidt med i, hvornår der ryddes op.

Lav ny klasse

1. Lav et projekt i Visual Studio til ny C# konsol-applikation
 - a. Kald projektet "ObjectCounter" og
 - b. Placer det i
C:\Wincubate\87356\Module 10\Lab 10.2\Starter
2. Tilføj en ny C#-fil med navn "A.cs" og definér en klasse A i denne fil
 - a. Lav en `public static int` property på A og kald denne `InstanceCount`
 - i. Lav `set` private
 - b. Lav en statisk constructor, der sætter `InstanceCount` til 0
 - c. Definér en default constructor, som tæller `InstanceCount` én op
 - d. Definér en destructor, som tilsvarende tæller `InstanceCount` én ned
 - i. På denne måde vil `InstanceCount` hele tiden indeholde antallet af A-instanser i applikationen
3. Når du er færdig, bør resultatet ligne

```
class A
{
    public static int InstanceCount { get; private set; }

    static A()
    {
        InstanceCount = 0;
    }

    public A()
    {
        InstanceCount++;
    }

    ~A()
    {
        InstanceCount--;
    }
}
```

4. Byg programmet og korriger eventuelle fejl.

Generér objekter

5. I `Program.cs` skal `Main()` skrives, så
 - a. Der laves en uendelig `while`-løkke indeholdende
 - i. Udskrift af teksten
Press ENTER to generate objects
 - ii. En efterfølgende `Console.ReadLine()`, der venter på tryk på Enter

iii. En efterfølgende for-løkke, der genererer 10000 instanser af A, som med det samme er i spil til at blive garbage collected

1. Lav en lokal instans i hver iteration af for-løkken

iv. Udskrift af teksten

There are currently n instances of A
hvor n fås fra InstanceCount.

6. Når du er færdig, bør resultatet ligne

```
static void Main(string[] args)
{
    while (true)
    {
        Console.WriteLine("Press ENTER to generate objects");
        Console.ReadLine();

        for (int i = 0; i < 10000; i++)
        {
            A a = new A();
        }

        Console.WriteLine("There are currently {0} instances of A",
            A.InstanceCount);
    }
}
```

7. Byg programmet og korriger eventuelle fejl.

8. Kør programmet og tryk Enter nogle gange, mens du observerer resultaterne.

9. Find forklaringen på, hvad det er, der sker

b. Hvad sker der med antallet af A-instanser?

c. Kan I mon observere noget mærkeligt...? ☺

Øvelse 10.3: "IComparer på Card" (★)

Man kan implementere forskellige sorteringer ved at lave flere forskellige små hjælpeklasser, som implementerer IComparer og så bruge instanser af disse til at give med til `Array.Sort()` som f.eks.

```
Array.Sort( _cards, new RankComparer() );
```

Opgaven er nu som følger:

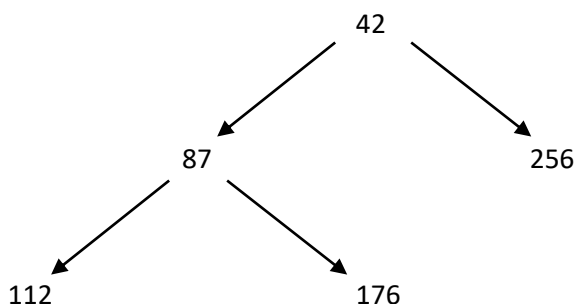
- Tag udgangspunkt i løsningen på Øvelse 10.1, der er at finde i `C:\Wincubate\87356\Module 10\Lab 10.3\Starter`
- Lav en `RankComparer`-klasse, der implementerer `IComparer` og sørger for, at ovenstående linie vil sortere kortene efter værdi først og kulør bagefter.
- Lav en metode på `Deck`, `ArrangeByRank()`, der benytter denne teknik.
- Kør programmet og check, at programmet gør som du forventer.

Hvis der er tid, og det skal være ekstra pænt, kan man tænke over, at `RankComparer` kun giver mening for `Card`. Så man burde

- Lave en statisk property på `Card`, der returnerer en ny `RankComparer`, når den skal bruges...

Øvelse 10.4: "IEnumerable på rekursive datatyper" (☆☆☆)

I denne opgave vil vi implementere `IEnumerable` på den rekursive datatyper, som vi definerede i Øvelse 7.3, hvor vi skrev en klasse `Node`, der selv kan referere instanser af `Node`. `Node` udgjorde knuderne i et binært træ, som for eksempel



Hver af knuderne i træet har en heltalsværdi repræsenteret ved en `int`. (Se evt. Øvelse 7.3 for flere detaljer)

- Tag udgangspunkt i løsningen fra Øvelse 7.3, der er at finde i `C:\Wincubate\87356\Module 10\Lab 10.4\Starter`.
- Fjern metoden `Print()` og implementér `IEnumerable` på en snedig måde, så ovenstående træ enumereres som sekvensen 42, 87, 112, 176, 256.
- Test din implementation ved at ændre `Main()` til at udskrive node vha. en `foreach`-sætning.
- Fjern tilsvarende metoden `PrintPostfix()` og implementér `IEnumerable` på en snedig måde, så ovenstående træ enumereres som sekvensen 112, 176, 87, 256, 42
 - **Bemærk:** Da der nu er to forskellige implementationer af iteratorer til `IEnumerable` på `Node`, er det nødvendigt nu at benytte "**named** iterators".
- Test din implementation ved at ændre `Main()` til at også at udskrive node vha. en `foreach`-sætning.

Module 11: “Collections and Generics”

Øvelse 11.1: “Brug generiske collections”

Denne opgave går ud på at benytte generiske collection-klasser i et lille eksempel.

I programskabelonen, der er at finde i

`C:\Wincubate\87356\Module 11\Lab 11.1\Starter`

findes der et simpelt program, som indlæser strenge fra Console indtil en tom streng indlæses.

- Inspicér programmet og indse, hvordan det virker.
- Udfyld de manglende programstumper
 - Erklær og initialisér først en generisk streng-liste
 - Indsæt derefter hver indlæst streng i streng-listen
 - Iterér nu gennem streng-listen og indsæt de tilsvarende længder på strengene i en generisk heltals-kø, som skal oprettes til formålet
 - Iterér nu gennem heltals-køen og skriv hvert tal ud
 - Beregn summen af alle de indgående tal og skriv dette ud til sidst.

Øvelse 11.2: "Brug Dictionary<K,V>"

Denne opgave går ud på at benytte Dictionary-klassen.

Lav et nyt projekt i

C:\Wincubate\87356\Module 11\Lab 11.2\Starter

hvor du benytter Dictionary<int, string> til at holde en afbildning af positioner i dagens lokale til fornavnene på de personer, der sidder på dem.

- Lav en instans af Dictionary<int, string>.
- Initialiser dictionaryet, så det indeholder information som repræsenterer data på formen
 - 1 -> "Charlotte"
 - 3 -> "Tim"
 - 4 -> "Anne-Mette"
 - 6 -> "Karl Erik"
- Skriv nu en foreach-løkke, der udskriver ovenstående information på formen

A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window has a black background with white text. The text displayed is: "Charlotte sidder på plads 1", "Tim sidder på plads 3", "Anne-Mette sidder på plads 4", "Karl Erik sidder på plads 6", and "Press any key to continue . . .". The window has standard Windows window controls (minimize, maximize, close) in the top right corner and a scrollbar on the right side.

Øvelse 11.3: "Lav generisk Singleton-klasse" (★)

"Design patterns" er klassiske, velkendte og genbrugbare løsninger på ofte forekommende problemer. Ét af disse er Singleton-patternet, der går ud på at sikre, at der altid højst forekommer én "global" instans af en given klasse.

I denne øvelse vil vi se, hvordan generics passer perfekt til at lave en genbrugbar implementation af Singleton, som I kan tage med hjem og putte i jeres C# værktøjskasse.

Implementér Singleton<T>

1. Lav et nyt projekt til en konsolapplikation i
C:\Wincubate\87356\Module 11\Lab 11.3\Starter.
 - a. Kald det "SingletonPattern".
2. Tilføj en ny klasse i en ny fil "Singleton.cs"
 - a. Lav en statisk klasse Singleton<T>, hvor
 - i. T kræves at være en reference-type
 - ii. T kræves at have en default constructor.
 - b. Lav inde i Singleton<T> en statisk member variabel af type T, der hedder _instance.
 - i. Lav _instance som private
 - ii. Initialiser _instance til null som initial-værdi
 - c. Lav på klasse Singleton<T> en public statisk property af type T, der hedder Instance
 - i. Lav en tilhørende get-metode, der
 1. hvis _instance er null, sætter _instance til en ny instans af T
 2. Returnerer _instance.
3. Når du er færdig, bør resultatet ligne

```
public static class Singleton<T> where T : class, new()
{
    private static T _instance = null;

    public static T Instance
    {
        get
        {
            if( _instance == null )
            {
                _instance = new T();
            }
            return _instance;
        }
    }
}
```

4. Byg programmet og korriger eventuelle fejl.

Benyt Singleton<T>

5. Tilføj en ny klasse i en ny fil "TestClass.cs"
 - a. Lav en public klasse TestClass

- b. Lav en public default constructor for TestClass, som skriver ud på Console, at constructoren er kaldt
 - c. Lav på TestClass en public property af type int, der hedder X med både get og set
 - d. Lav på TestClass en public metode, Display(), der hverken modtager parametre eller returnerer noget, men blot skriver værdien af X ud på Console.
6. Når du er færdig, bør resultatet ligne

```
public class TestClass
{
    public int X
    {
        get;
        set;
    }

    public TestClass()
    {
        Console.WriteLine( "TestClass.TestClass()" );
    }

    public void Display()
    {
        Console.WriteLine( "X = {0}", X );
    }
}
```

7. Byg programmet og korriger eventuelle fejl.
8. Udfyld nu Main()-metoden i "Program.cs" til at teste din Singleton-implementation med TestClass
- e. Benyt Singleton<TestClass> til at få en "global" instans, t1, af TestClass
 - f. Sæt X på t1 til 42
 - g. Benyt Singleton<TestClass> til at få en (ny?) "global" instans, t2, af TestClass
 - h. Kald Display() på t2.
9. Når du er færdig, bør resultatet ligne

```
static void Main(string[] args)
{
    TestClass t1 = Singleton<TestClass>.Instance;
    t1.X = 42;

    TestClass t2 = Singleton<TestClass>.Instance;
    t2.Display();
}
```

10. Byg programmet og korriger eventuelle fejl.
11. Kør programmet! Hvad sker der?
- i. Hvad er ligheden mellem de to instanser lavet med Singleton?
 - j. Hvor mange gange kaldes constructoren på TestClass?
 - k. Hvilke klasser virker dette for...?
 - l. Hvad sker der mon, hvis vi havde testet Singleton<T> med en værdi-type TestStruct?

Bemærk: Opbygningen af `Singleton<T>` ses ofte i mange forskellige afskygninger indeholdende nogle sindrige detaljer og ekspert-indsigt, som er resultatet af mange folks årelange diskussioner – herunder omkring trådsikkerhed af klassen. En god sammenfatning er at finde på <http://www.yoda.arachsys.com/csharp/singleton.html> som kan læses, hvis man engang skal lave multitrådede programmer 😊

Øvelse 11.4: "Sorterede mængder" (☆☆)

De nye mængde-collections introduceret i .NET 4.0 er ofte også særdeles brugbare. I denne opgave vil vi benytte dem til at finde og sortere mængder af personnavne.

Datagrundlaget for opgaven er givet i start-projektet i form af en `List<Person>`, der fra start indeholder en masse instanser af typen `Person`.

Sorteret mængde forekommende fornavne

- Tag udgangspunkt i navne-datagrundlaget, der er at finde i `C:\Wincubate\87356\Module 11\Lab 11.4\Starter`.
- Lokalisér `TODO 1` i `Program.cs`.
- Lav et stykke kode til at beregne mængden af fornavne, der forekommer i data-grundlaget, og udskriv dem sorteret alfabetisk.

Forekommende efternavne sorteret efter længde

- Lokalisér `TODO 2` i `Program.cs`.
- Udvid din løsning ovenfor til at beregne mængden af efternavne, der forekommer i data-grundlaget, og udskriv dem sorteret efter længde og derefter alfabetisk.
 - Hint: Lav en klasse `LastNameComparer`, der implementerer `IComparer<T>` og lav et `SortedSet<T>` vha. denne.

Module 12: “Delegates, Events, and Lambda Expressions”

Øvelse 12.1: “Delegates”

Denne øvelse laver et par eksempler for at illustrere, hvorledes delegates er referencer til metoder, som kan kaldes præcis som metoder.

Delegates til heltalsfunktioner

1. Lav et nyt projekt “DelegateFun” i
C:\Wincubate\87356\Module 12\Lab 12.1\Starter.
2. Definér en delegate-type ved navn `IntAction`, som kan referere metoder, der tager et heltal som parameter og returnerer `void`.
3. Åbn den eksisterende “Program.cs” fil
 - Lav en statisk metode `PrintInt()` med en signatur, som matcher `IntAction`
 - Metoden skal skrive det medfølgende argument ud på Console
 - Lav ydermere en statisk metode `DoForAll()`, der tager to parametre af typerne `IntAction` og `int[]`, henholdsvis, og returnerer `void`
 - Metoden skal kalde den medfølgende delegate på hvert heltal i arrayet.
4. Test dine implementationer ved at lave et array bestående af tallene 42, 87, 112, 176, 256 og print tallene i dette array ud vha. `DoForAll()`.

Øvelse 12.2: "Aktier og events"

Denne opgave går ud på at benytte publishers og subscribers af aktie-kurser, der kommunikerer vha. events. Vi vil først konstruere en StockChanged event for en publisher med tilhørende event-argumenter, for derefter at implementere en subscriber til disse events.

Lav StockChangedEventArgs

1. Åbn det eksisterende projekt "Stocks" i
C:\Wincubate\87356\Module 12\Lab 12.2\Starter.
 - a. Verificér, at projektet i "StockPublisher.cs" indeholder en skabelon til en klasse kaldet StockPublisher, som senere skal udfyldes nedenfor.
 - i. Læg mærke til member-variablen `_ticker`, der indeholder navnet på aktien.
 - ii. Inspicér kort resten af klassen, der sørger for, at metoden `OnStockChanged()` kaldes med jævne mellemrum med en ny aktiekurs
 1. Disse detaljer forventes I ikke at forstå i dybden!
 - b. Verificér, at projektet i "Program.cs" indeholder den sædvanlige `Main()`-metode.
2. Tilføj en ny klasse i en ny fil "StockChangedEventArgs.cs"
 - a. Lav en public klasse `StockChangedEventArgs`, der arver fra `EventArgs`
 - b. Lav på klassen en public automatisk property af type `string`, der hedder `Ticker`
 - i. Den skal være readonly: Kun med en `get`;
 - c. Lav på klassen en public automatisk property af type `double`, der hedder `StockValue`
 - i. Den skal være readonly: Kun med en `get`;
 - d. Lav på klassen en public automatisk property af type `DateTime`, der hedder `TimeStamp`
 - i. Den skal være readonly: Kun med en `get`;
 - ii. Lav en initializer, der sætter initial-værdien til det aktuelle tidspunkt.
 - e. Lav en public constructor for `StockChangedEventArgs`, der tager to parametre `ticker` og `stockValue` og sætter de tilsvarende properties
3. Når du er færdig, bør resultatet ligne

```
public class StockChangedEventArgs : EventArgs
{
    public string Ticker { get; }
    public double StockValue { get; }
    public DateTime TimeStamp { get; } = DateTime.Now;

    public StockChangedEventArgs( string ticker,
                                double stockValue )
    {
        Ticker = ticker;
        StockValue = stockValue;
    }
}
```

4. Byg programmet og korriger eventuelle fejl.

Implementér StockPublisher

5. Åbn nu filen "StockPublisher.cs".
6. Lokalisér "TODO: Define StockChanged event"

- a. Definér en public event kaldet StockChanged, som man kan subscribe til med delegates af typen EventHandler<StockChangedEventArgs>
- 7. Lokalisér "TODO: Raise StockChanged event"
 - b. Lav nu kode til at fyre StockChanged eventen.
 - c. Check om StockChanged er forskellig fra null, og hvis den er det, skal den kaldes med event-argumenter indeholdende
 - i. Navnet på den underliggende aktie
 - ii. Den medfølgende stockValue
 - d. Brug evt. den nye ?.Invoke() til at gøre dette.
- 8. Når du er færdig, kan resultatet ligne

```
// TODO: Define StockChanged event
public event EventHandler<StockChangedEventArgs> StockChanged;

protected void OnStockChanged(double stockValue)
{
    // TODO: Raise StockChanged event
    StockChanged?.Invoke(
        this, new StockChangedEventArgs( _ticker, stockValue )
    );
}
```

- 9. Byg programmet og korriger eventuelle fejl.

Implementér StockSubscriber

- 10. Tilføj en ny klasse i en ny fil "StockSubscriber.cs"
 - a. Lav en public klasse StockSubscriber
 - b. Lav på klassen en public metode, der hedder SubscribeTo(), som returnerer void og som parameter tager en instans af StockPublisher
 - i. Sørg for, at denne metode subscriber på StockChanged-eventen på den medfølgende StockPublisher ved at kalde en ny metode, som du laver nedenfor
 - c. Lav på klassen en private metode OnStockChanged(), der matcher signaturen på StockChanged-eventen,
 - i. Udskriv på Console inde i metoden nedenstående streng
Stock X was on Y priced at Z
 - ii. Den underliggende aktiekurs ønskes udskrevet med to decimaler.
 - iii. Brug evt. den nye \$-operator på strenge til dette.
- 11. Når du er færdig, kan resultatet ligne

```

public class StockSubscriber
{
    public void SubscribeTo(StockPublisher p)
    {
        p.StockChanged += OnStockChanged;
    }

    private void OnStockChanged(object sender,
                                StockChangedEventArgs e)
    {
        Console.WriteLine(
            $"Stock {e.Ticker} was on {e.TimeStamp} priced at {
e.StockValue:f2}" );
    }
}

```

12. Byg programmet og korriger eventuelle fejl.

Sæt alle elementerne sammen

13. Åbn nu filen "Program.cs".

14. Lokalisér "TODO: Create publishers and subscriber"

- a. Lav en StockPublisher med aktienavn "MSFT"
- b. Lav en StockPublisher med aktienavn "WCB"
- c. Lav en StockSubscriber, der abonnerer på de to StockPublisher-instanser
- d. Tilføj til sidst

```
Console.ReadLine();
```

så programmet ikke stopper med den samme

15. Når du er færdig, bør resultatet ligne

```

// TODO: Create publishers and subscriber
StockPublisher p1 = new StockPublisher("MSFT");
StockPublisher p2 = new StockPublisher("WCB");

StockSubscriber subscriber = new StockSubscriber();
subscriber.SubscribeTo(p1);
subscriber.SubscribeTo(p2);

Console.ReadLine();

```

16. Byg programmet og korriger eventuelle fejl.

17. Kør programmet

- e. Hvad ser du?

Variér konceptet

18. Prøv at udvide programmet med flere subscribers og flere publishers

- a. Hvad sker der nu, når du kører?

19. Hvad sker der mon, hvis du kalder SubscribeTo() flere gange på samme StockSubscriber med samme StockPublisher...? Prøv! ☺

Øvelse 12.3: "Anonyme metoder og lambda-udtryk" (☆☆)

Et prædikat er en afbildning, som tager et argument af en given type og returnerer en boolean, der angiver om argumentet opfylder prædikatet. I .NET findes indbygget en generisk delegate-type `Predicate<T>`, der er et prædikat af type `T`.

Tilsvarende findes der på `Array`-klassen en statisk, generisk metode `FindAll()`, der som parametre tager et array af type `T` samt et `Predicate<T>` og returnerer et array af bestående af netop de elementer, som opfylder prædikatet.

Tag udgangspunkt i et array defineret ved

```
string[] names = { "Kim", "Mads", "Rasmus", "Bo", "Jesper" };
```

Opgaven er nu – på tre forskellige måder – ved hjælp af prædikater og `Array.FindAll()` at udskrive alle navne i listen med mindst 4 bogstaver:

1. Først vha. en metode, der matcher `Predicate<T>`.
2. Dernæst ved brug af en anonym metode.
3. Slutteligt – og mest elegant – vha. et passende lambda-udtryk.

Til sidst:

4. Kan man lave et lambda-udtryk, som skriver navnene ud mens den filtrerer, så vi ikke behøver skrive det resulterende array ud bagefter? Hvordan...?

(Husk at få en tåre i øjenkrogen af beundring, når du har fundet løsningen. 😊)

Module 13: “Advanced C# Language Features”

Øvelse 13.1: “Indexere på gode, gamle Deck” (★)

Formålet med denne øvelse er at illustrere, hvorledes man kan introducere indexere på kortspilstypen, som vi har stiftet bekendtskab med nogle gange gennem kurset.

- Tag udgangspunkt i løsningen fra Øvelse 11.2, der er at finde i
C:\Wincubate\87356\Module 13\Lab 13.1\Starter
 - Verificér, at det er Deck, som vi kender den
 - Opfrisk kort, hvordan tingene fungerer.

Implementér en heltals-indexer på Deck

- Lav i filen “Deck.cs” en heltals-indexer på Deck, der
 - har både en `get`; og `set`;
 - tager et indeks af typen `int` kaldet `index`
 - returnerer en værdi af type `Card`
 - Den værdi, som findes på indeks `index` i kortspillet
 - kaster en `IndexOutOfRangeException` med en passende informativ tekst, hvis det anvendte indeks er udenfor det underliggende array af kort.

Implementér en streng-indexer på Deck

- Lav ligeledes i filen “Deck.cs” en streng-indexer på Deck, der
 - kun har en `get`;
 - tager et indeks af typen `string` kaldet `s`
 - returnerer en værdi af type `int`
 - Det indeks i kortspillet (hvis det findes), hvor `Card.ToString()` er netop `s`
 - kaster en `IndexOutOfRangeException` med en passende informativ tekst, hvis det anvendte indeks ikke findes i det underliggende array af kort.

Test dine implementationer

- Test din implementation ved at tilføje kode til `Main()`, der tester begge indexers ovenfor.

Øvelse 13.2: "Extension-metoder"

Denne øvelse går ud på at definere to extension-metoder på `int` og `DateTime`, hhv., for på den måde at bygge genbrugbare metoder, som I kan udvide jeres værktøjskasse med.

Implementation af extension-metoder

1. Lav et nyt projekt "ExtensionsMethods" i
C:\Wincubate\87356\Module 13\Lab 13.2\Starter.
2. Lav en ny fil "Extensions.cs"
 - Lav namespace i filen om til `<Jeres_firmanavn>.Utility`
 - Eksempelvis `Wincubate.Utility`.
 - Implementér i denne fil en extension-metode `IsEven()` på `int`, som returnerer en `bool`, der angiver om tallet er lige.
 - Implementér ligeledes i denne fil en extension-metode `To<Jeres_firmanavn>TimeStamp()` på `DateTime`, som returnerer en streng med jeres yndlingsmåde at lave timestamps på
 - Eksempelvis som vist i slidesene.

Test af extension-metoder

3. Åbn den eksisterende "Program.cs" fil og lav kode til at teste jeres ny-implementerede metoder.
4. Hvad sker der, når I prøver at kalde metoderne?
5. For at I kan kalde extension-metoderne, skal de "bringes ind i scope" – Enten vha. at benytte hele stien til metoden eller at benytte
`using <Jeres_firmanavn>.Utility;`
6. Prøv nu at kalde metoderne på udvalgte heltal og `DateTime`.
 - Virker det nu?
 - Hvordan ser Intellisense ud for metoderne?

Hvis du vil se flere extension-metoder...

7. Find en spændende extension-metode fra <http://www.extensionmethod.net> og afprøv den 😊

Øvelse 13.3: “Object initializers” (☆☆)

I denne øvelse vil vi – som opvarmning til Module 14 – skrive et større data-udtryk som en anonym type vha. de såkaldte object initializers.

1. Betragt typerne

```
enum CustomerCity { Aarhus, Horsens }

class Customer
{
    public string Name { get; set; }
    public CustomerCity City { get; set; }
    public List<Order> Orders { get; set; }
}

class Order
{
    public int Quantity { get; set; }
    public Product Product { get; set; }
}

enum ProductName { Mælk, Smør, Brød, Øl, PepsiMax }

class Product
{
    public ProductName Name { get; set; }
    public double Price { get; set; }
}
```

der alle er at finde i projektet i

C:\Wincubate\87356\Module 13\Lab 13.3\Starter .

2. I samme projekt findes et udtryk af type List<Customer> lavet vha. object initializers, der består af nedenstående data

- Kunde: "Kim" fra Aarhus
 - 3 stk. ordrer: Mælk, Smør, Brød
- Kunde: "Mads" fra Horsens
 - 4. stk. ordrer: Mælk, Smør, Brød, Øl
- Kunde: "Jesper" fra Aarhus
 - 1 stk. ordre: Pepsi Max

(med en relevant pris for hver...)

3. Opgaven er nu at

- Forstå indeholdet og effekten af den allerede eksisterende kode.
- Konvertere det derefter til et lignende udtryk konstrueret udelukkende vha. anonyme typer
 - Hvordan er fremgangsmåden for dette?
 - Brug var-keywordet, hvor nødvendigt.
- Skrive et udtryk, der udskriver alle (de anonymt typede) kunder, ordrer og produkter.

Module 14: “LINQ to Objects”

Øvelse 14.1: “Kunder og ordrer i LINQ”

Vi vil i denne opgave beskæftige os med grundlæggende queries i LINQ. Øvelsen tager udgangspunkt i data-grundlaget fra Øvelse 13.4, hvor vi skabte et initialiserende udtryk bestående af Customer, Order og Product objekter til brug netop her.

1. Åbn skabelonen med ovennævnte data alle er at finde i projektet i
C:\Wincubate\87356\Module 14\Lab 14.1\Starter .

Udvælg alle kunder med navn og by

2. Lokalisér ”TODO 1: All customers name and city”.
3. Skriv en LINQ query, der udvælger alle kunder i customers .
4. Udskriv vha. ovenstående query for hver kunde deres navn og by.
5. Byg og kørs programmet
 - Test af din query er korrekt.

Udskriv alle kunder fra Aarhus med navn

6. Lokalisér ”TODO 2: All customers from Aarhus by name”.
7. Udskriv vha. en passende LINQ query alle de kunder i customers, som kommer fra Aarhus

Udskriv antallet af ordrer for Kim

8. Lokalisér ”TODO 3: Number of orders placed by Kim”.
9. Udskriv vha. en passende LINQ query antal order for Kim.

Øvelse 14.2: "LINQ Queries og extension-metoder" (★)

Ofte kan det være fordelagtigt at kombinere de gængse LINQ-operatorer med udvalgte LINQ extension-metoder til at konstruere sammensatte resultater. Andre gange kan det være lettere at benytte extension-metoderne hørende til de kendte LINQ-keywords. Som regel er valget baseret på smag og behag,.

Som data-grundlag i denne opgave vil vi benytte listen af spilnavne til Wii og Xbox 360.

1. Åbn skabelonen med det beskrevne data-grundlag i projektet i
C:\Wincubate\87356\Module 14\Lab 14.2\Starter .

Udskriv alle Wii-spil sorteret efter titel

2. Lokalisér "TODO 1: All Wii games sorted by title".
3. Udskriv vha. en passende LINQ query alle Wii-spil sorteret alfabetisk efter titel.

Udskriv alle Wii-spil sorteret efter titel-længde med længste først

4. Lokalisér "TODO 2: All Wii games sorted by title length (longest title first)".
5. Udskriv vha. en passende LINQ query alle Wii-spil aftagende efter titel-længde.

Udskriv med store bogstaver alle spil på enten Wii eller Xbox

6. Lokalisér "TODO 3: All games for either machine sorted by title".
7. Udskriv vha. en passende LINQ query med store bogstaver alle spil til enten Wii eller Xbox 360 (uden dubletter!)

Udskriv alle spil til Wii, men ikke til Xbox, dog på nær dem med "Wii"

8. Lokalisér "TODO 4: All games for Wii but not for Xbox 360 except those with 'Wii' in the title".
9. Udskriv vha. en passende LINQ query de spil som findes til Wii men ikke Xbox – dog fraregnet dem, som indeholder "Wii" i titlen.

Øvelse 14.3: "Flere kunder og ordrer i LINQ" (☆☆)

Denne øvelse er en fortsættelse af Øvelse 14.1, hvor strukturen på de indgående queries dog er en smule mere komplicerede.

- Åbn skabelonen med de velkendte kunde-data i projektet i
C:\Wincubate\87356\Module 14\Lab 14.3\Starter .

Udskriv alle kunder der har købt mælk

- Udskriv vha. en passende LINQ query navnene på alle de kunder, som har købt mælk
 - Hint: Brug to from-clauses.

Udskriv total forbrugt ordresum

- Udskriv vha. en passende LINQ query den totale sum brugt på ordrene for alle kunder tilsammen
 - Hint: Der findes en extension-metode, der hedder Sum().

Udskriv forbrugt ordresum for hver kunde individuelt

- Udskriv vha. en passende LINQ query hver individuelle kundes sum brugt på vedkommendes ordrer
 - Hint: Her kan det være en god idé at introducere en lokal "hjælpe-variabel" i et udtryk vha. LINQ's let-konstruktion, f.eks.
`let V = from X in Y select Z.`

Udskriv kunder grupperet pr. by

- Udskriv vha. en passende LINQ query alle kunder grupperet efter, hvilken by de kommer fra
 - Hint: Her kan det være en god idé benytte group-by konstruktionen i LINQ, f.eks.
`from X in Y group X by Z.`