# Bank Account and Sorting

|  |  |
| --- | --- |
| **Læringsmål** | Du kan:   * **1Pf1:** anvende centrale metoder til at specificere og konstruere algoritmer [...] * **1Pf2:** anvende centrale faciliteter i programmeringssproget til realisering af algoritmer […] * **1Pf3**: anvende et i professionen udbredt, integreret udviklingsværktøj, herunder versionsstyringssystem […] til at designe og konstruere praksisnære applikationer […] * **1Pk3**: i en struktureret sammenhæng tilegne sig ny viden, færdigheder og kompetencer inden for programmeringssprog, udviklingsværktøjer, programmeringsteknikker og programdesign * **1Pf6**: anvende moderne teknikker og værktøjer til afvikling af test […] |
| **Forventet læringsudbytte [SOLO]** | Programmeringssporet:   * [Unistrukturel] Du kan huske enkelte kendetegn ved **programflow** – sekvens, forgrening (if-else, switch), løkke (while, do-while, for, foreach), metodekald/metoderetur, exception handling (try-catch-finally), jump statements (break, continue) * [Unistrukturel] Du kan genkende enkelte kendetegn ved **datatyper** - int, string, bool, double, char, array (Sort) * [Unistrukturel] Du kan genkende enkelte kendetegn ved **C# klasse** – C# klasse, felt, metode, property (get, set, backing field), auto-implemented properties, access modifiers (public, private), access keywords (this)   Færdighedssporet:   * [Unistrukturel] Du kan nævne enkelte kendetegn ved **testing** – unit test attributter (TestClass, TestMethod, TestInitialize), unit test assertions (Assert, CollectionAssert) * [Unistrukturel] Du kan huske enkelte kendetegn ved **debugging** –break point, step into, step over |
| **Din forberedelse** | Programmeringssporet:   * [Properties](https://csharp.net-tutorials.com/classes/properties/) * [Sort Array in C# without using inbuilt function](https://www.c-sharpcorner.com/blogs/sort-array-in-c-sharp-without-using-inbuilt-function) * [Using foreach with arrays](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/using-foreach-with-arrays) * [How to override the ToString method](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/how-to-override-the-tostring-method) * Properties – auto-implemented & side effects.mp4 (video: 29:17) |

I dagens opgave skal du bl.a. styre bevægelser til og fra en bankkonto via forskellig brug af properties (også auto-implementerede) samt metoder; du skal også arbejde med constructor chaining i samme case. Derudover introduceres du til den fjerde og sidste løkke-type, foreach-løkken, som du skal anvende i din implementering af en sorteringsalgoritme for heltalsarrays.

I begge tilfælde skal du anvende unit tests og debugging for at tjekke, om din implementering er korrekt.

# Dagens ord: Forløse frustration og forståelse af flow

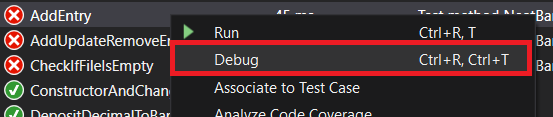
Du har måske oplevet en eller flere af de følgende frustrationspunkter:

* ”Hjælp, jeg forstår ikke, hvorfor min metode ikke returnerer det resultat, som jeg forventede”
* ”Hjælp, min løkke virker ikke, som den skal”
* ”Hjælp, jeg kan ikke forstå den kode, som vi har udarbejdet i gruppen”
* ”Hjælp, jeg kan ikke forstå den kode, som jeg har fået udleveret”
* ”Hjælp, jeg kan ikke forstå den kode, som jeg har fundet på [stackoverflow](https://stackoverflow.com/)”

Et værktøj, som du kan benytte til at forløse denne frustration og til at opnå en personlige forståelse af selve programflowet, er: **Debugging**

***Hvad så med unit tests?***

Ja, hvad så, hvis du har modtaget en række af unit tests, og du kan se, at din implementering fejler, og at du ikke modtager det forventede resultat? Kan du så benytte debugging her? Ja! Åbn Test Explorer, højreklik på den unittest, der fejler, og vælg ”Debug”, som på billedet nedenunder:



# Øvelse 1: Terminologi

Nu skal du teste din forforståelse af dagens emne(r) med udgangspunkt i dagens forberedelse.

Del teamet op i mindre grupper (af 2-3 personer), og brug **Ordet rundt** til at diskutere begreberne ”property”, ”auto-implemented property”, ”foreach loop”, ”sorting”, ”override”, ”constructor overloading”. Sørg for, at alle får mulighed for at tale.

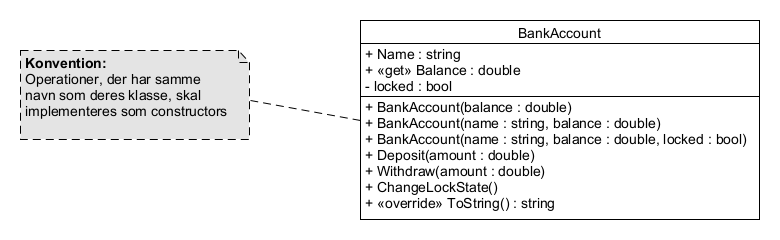
*Tidsramme: 20 minutter*

# Øvelse 2: Bankkonto

Du skal i denne øvelse udvikle en bankkonto, hertil skal du tage udgangspunkt i et DCD og en række af krav til implementeringen.

## Øvelse 2.1: Inspektion af DCD

Du og din sidemakker skal inspicere nedenstående DCD:



Overvej følgende, inden du går videre:

* Hvordan kan det være, at ”BankAccount” kan stå der flere gange under operationer? Hvorfor er det tilladt?
* Hvordan kan man vide, om attributten *Name* skal implementeres som et offentligt felt (en: public field), eller som en property med et eksplicit backing field eller som en auto-implemented property?
* Hvad betyder *‹‹override››*?
* Hvad betyder *‹‹get››*?

## Øvelse 2.2: Inspektion af krav til implementering

Du og din sidemakker skal inspicere følgende liste af krav til implementeringen:

* *ChangeLockState()*: ændrer den nuværende status (locked) til det modsatte af, hvad den er
* *ToString() : string*: returnerer en tekst-streng med navn og balance, som således:
  + Name: Bob, Balance: 250
* *Withdraw(amount : double):* trækker det angivende beløb (en: amount)
  + må ikke trække flere penge, end der er på bankkontoen
  + må ikke trække penge fra en låst bankkonto (en: locked)
* *Deposit(amount : double)*: indsætter det angivende beløb
  + må ikke være i stand til at indsætte til en låst bankkonto

Derudover skal der benyttes constructor chaining (genlæs eventuelt [forberedelsen](https://www.geeksforgeeks.org/c-sharp-constructor-overloading/) fra Ex12).

## Øvelse 2.3: Implementering

Nu er du og din sidemakker klar til at kode.

Udfør følgende:

* Opret en ny konsolapplikation med projektnavnet ”SimpleBank”
* Implementér BankAccount-klassen i jeres konsolapplikation ud fra ovenstående software designklasse (Øvelse 2.1) og krav til implementeringen (Øvelse 2.2)

## Øvelse 2.4: Test implementering

Det er nu tid til at teste, om implementeringen af BankAccount-klassen er korrekt.

Udfør følgende:

* Opret et unittest projekt (MSTest Test Project) med navnet ”SimpleBankTest”
* Hent unittestene i materialefolderen for programmering (benyt: git pull). Du burde nu have en ny fil: *Ex14-BankAccountAndSorting.UnitTest.BankAccount.docx*
* Indsæt unittestene i dit testprojekt og sørg for, at testprojektet har en reference til konsolprojektet, samt at testen også kender til konsolprojektets namespace (using direktiv)
* Åbn Test Explorer, og kør unit-testene – få alle tests til at blive grønne
  + Det er i denne sammenhæng meget oplagt at benytte sig af debugging (genlæs eventuelt ”Dagens ord”)

# Øvelse 3: Loops

I denne øvelse skal du blive klogere på et par forskelle, der er mellem et udvalg af loops, som vi benytter os af, specifikt: while, for og foreach. Herefter skal du lave en manuel sortering af et array.

## Øvelse 3.1: Forstå syntaktisk sukker

Lav en inspektion af følgende kodeudsnit:

|  |
| --- |
| for (int i = 0; i < length; i++)  {  // do something  } |

Udfør herefter følgende:

* Benyt et while loop til at beskrive den samme logik, som der er i det ovenstående *for loop*

Del din løsning med din sidemakker.

## Øvelse 3.2: Foreach loop begrænsning

Lav en inspektion af følgende kodeudsnit:

|  |
| --- |
| int[] IntArray = { 3, 2, 4, 1 };  foreach (int number in IntArray)  {  number = 1;  } |

Overvej herefter følgende:

* Hvad sker der, hvis man benytter et foreach loop og ændrer i selve elementerne, som man er ved at iterere over? (som i eksemplet ovenover)

Afprøv din formodning:

* Opret en konsolapplikation
* Erklær et int array med navnet intArray
* Initialiser dit array med følgende tal: 3,2,4,1
* Benyt et foreach loop og itererer over dit array
  + inden i kodekroppen (scope) til dit loop, ændre da på et element i dit intArray

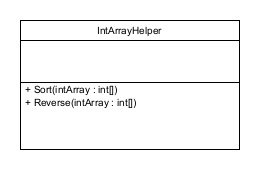
Gik det som du havde forestillet dig?

## Øvelse 3.3: Sortering med IntArrayHelper

Du skal i denne øvelse udarbejde en hjælper-klasse, der skal hjælpe med at sortere heltals arrays.

### Øvelse 3.3.1: Inspektion af software designklasse og krav til implementering

Lav en inspektion af nedenstående software designklasse:



Der er desuden følgende krav til implementeringen:

* *Sort(intArray : int[])*: sorterer et array af heltal, sorterer efter det mindste tal først og det højeste til sidst. Også kendt som en stigende sortering (en: ascending)
  + *F*.eks: 1, 2, 3, 4
* *Reverse(intArray : int[]):* sorterer et array af heltal, sorterer efter det højeste tal først og det laveste til sidst. Også kendt som en faldende sortering (en: descending)
  + F.eks: 4, 3, 2, 1

### Øvelse 3.3.2: Implementering af hjælper-klasse

Benyt parprogrammering

Udfør følgende:

* Opret en konsolapplikation med navnet SortingArrays
* Implementér ArrayHelper-klassen ud fra ovenstående software designklasse og krav til implementeringen (Øvelse 3.3.1)
  + Det er **ikke** tilladt at benytte *Array.Sort()* og *Array.Reverse()*

Hint: Hvis det driller med implementeringen, så er det oplagt at se på forberedelsesmaterialet (Hint: [Sort Array in C# without using inbuilt function](https://www.c-sharpcorner.com/blogs/sort-array-in-c-sharp-without-using-inbuilt-function)).

|  |
| --- |
| Bonusleg til at fremme algoritmisk forståelse: Det kan hjælpe at afprøve denne sortering ved at ”lege” med et stykke papir:   * Hiv et A4-papir i fire lige store stykker, og skriv ”1” på det første stykke papir, ”2” på det næste, ”3” på det tredje og ”4” på det sidste. * Overvej, hvordan du ville sortere disse tal manuelt og herefter, hvordan du kan beskrive dette som en algoritme. |

## Øvelse 3.4: Test implementering

Du skal nu tilføje et unittest projekt (MSTest Test Project) til din solution, herefter skal du hente unittestene i materialefolderen for programmering (benyt: git pull). Du burde nu have en ny fil: *Ex14-BankAccountAndSorting.UnitTest.SortReverseIntArray.docx.*

Kopiér indholdet ind i dit nyoprettede unittest projekt, og få alle tests til at blive grønne.

Overvej, om du har behov for at benytte debugging (*genlæs eventuelt ”Dagens ord”).*

Bemærk, at du med dit MSTest projekt benytter dig af det namespace, der hedder Microsoft.VisualStudio.TestTools.UnitTesting. Dette namespace tillader dig at benytte flere forskellige typer af Assert-klasser, herunder: Assert, CollectionAssert og StringAssert. I denne opgave benytter du en ny type af Assert, som du ikke har set på tidligere: CollectionAssert, for at se om samlingen af objekter i din samling er ens, herunder tjekkes selve tilstanden.