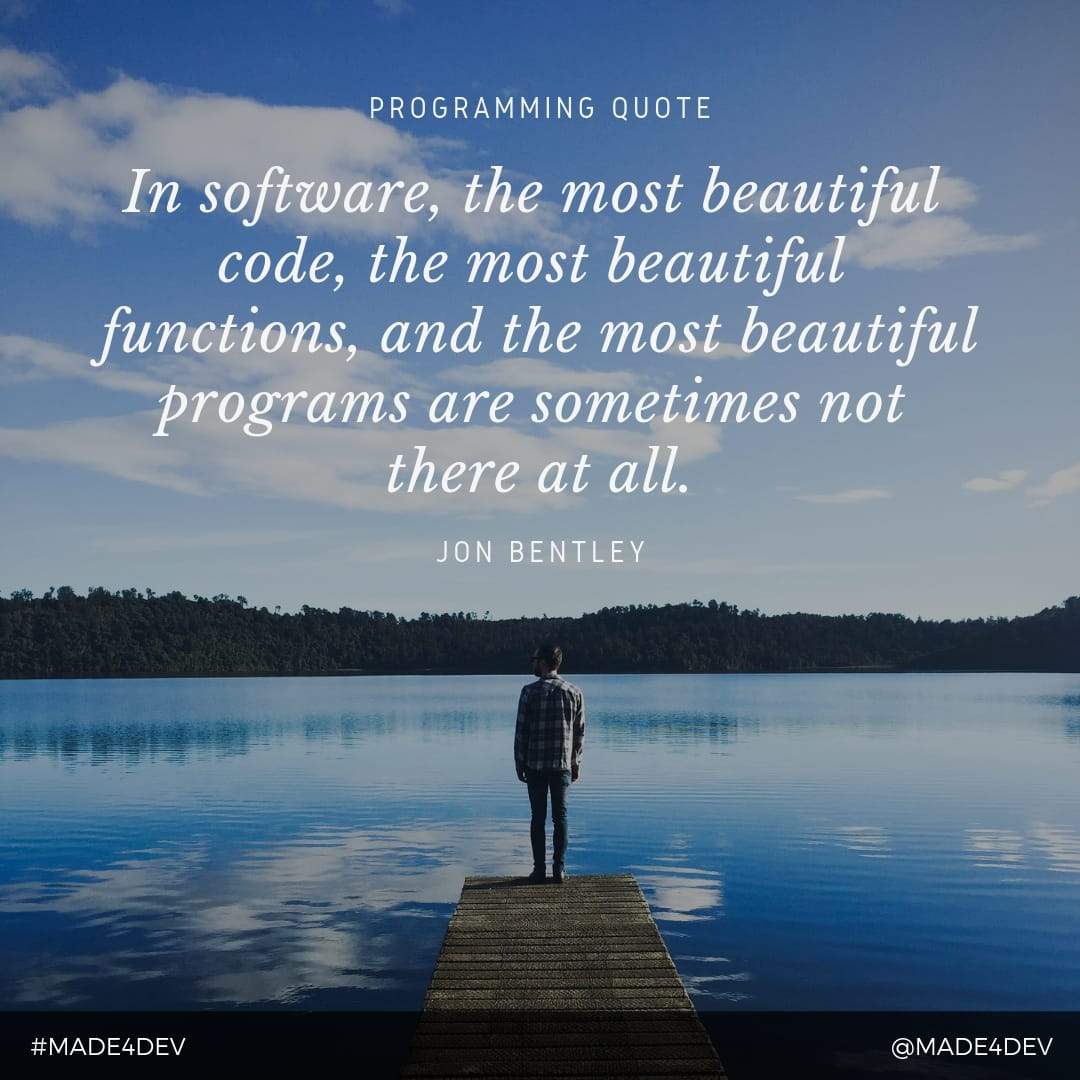
# Disaheim 2

|  |  |
| --- | --- |
| **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 styring af **program flow** * [Unistrukturel] Du kan huske enkelte kendetegn ved styring af **datahåndtering** * [Unistrukturel] Du kan genkende enkelte kendetegn ved simple **C# datatyper** * [Unistrukturel] Du kan genkende enkelte kendetegn ved **C# klasse (objekt)** –  […], constructor overloading, method overloading * [Unistrukturel] De kan nævne enkelte kendetegn i forhold til **OOP-principper** – indkapsling, polymorfi * [Unistrukturel] Du kan genkende enkelte kendetegn ved **.NET datatyper** – object-datatypen, value- & reference-typer, List<T> collections (da: samlinger)   Færdighedssporet:   * [Unistrukturel] Du kan huske enkelte kendetegn ved **testing** i Visual Studio IDE |
| **Din forberedelse** | Programmeringssporet:   * [Method Overloading](https://www.youtube.com/watch?v=TILr_yms5TA) (video: 7:17) * [C# Object Type Tutorial](https://www.koderhq.com/tutorial/csharp/object-type/) (læs til og med afsnittet ’Object arrays’) * [C# Value & Reference types](https://www.youtube.com/watch?v=eehaSv4vofA) (video: 3:26) * [ref vs out in C#](https://www.c-sharpcorner.com/UploadFile/ff2f08/ref-vs-out-keywords-in-C-Sharp/) * [C# Fundamentals: Working with Collections](https://channel9.msdn.com/Series/C-Fundamentals-for-Absolute-Beginners/22) (video: 33:59) * Gense SOLO: [Understanding understanding](https://www.youtube.com/watch?v=SfloUd3eO_M&feature=youtu.be&t=196) (video: fra 3:15 til 6.18) |

Du skal fortsætte med Disaheim-casen, hvor du gennem øvelserne skal arbejde med Collections (da: samlinger) i form af datatypen List<T>, som kan bruges i stedet for arrays, samt method overloading svarende til constructor overloading. Du kommer også til at arbejde med unit-tests og debugging. Du får også et par øvelser om forskellen på reference- og value-typer ved metodekald.

# Dagens ord:



# Øvelse 1: Terminologi

Del teamet op i to mindre grupper, og brug **Møde på midten** i hver gruppe til at diskutere begreberne ”value- og reference type”, ”Collections og List<T>” og ”Method overloading”.   
Husk, at alle i gruppen skal have taletid.

*Tidsramme: 20 minutter*

**Benyt parprogrammering til alle de resterende øvelser**.

# Øvelse 2: Disaheim - Økonomi

Formålet med Disaheim-systemet er at hjælpe virksomheden med at få et overblik over deres aktiviteter – herunder økonomi.

## Øvelse 2.1: Beregning af værdi

Du får derfor brug for at kunne beregne værdien af en bog og en amulet, som udtrykt i nedenstående operationskontrakter:

Operationskontrakt for **GetValueOfBook(book : Book) : double**

Cross reference: Budgetopfølgning

Precondition: Bogen anvendt som parameter eksisterer

Postcondition: Ingen ændringer i systemet

Output: Bogens værdi givet ved Price returneres

Operationskontrakt for **GetValueOfAmulet(amulet : Amulet) : double**

Cross reference: Budgetopfølgning

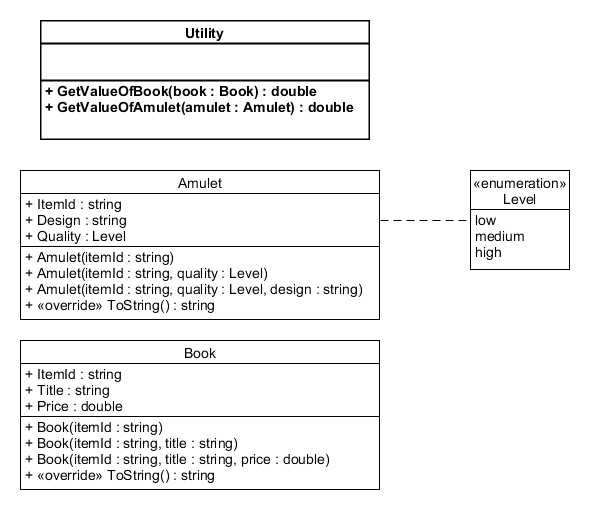
Precondition: Amuletten anvendt som parameter eksisterer

Postcondition: Ingen ændringer i systemet

Output: Amulettens værdi returneres.

Værdien fastsættes ud fra amulettens kvalitet:   
low -> 12.5, medium -> 20.0 og high -> 27.5

I DCDet placeres disse to operationer i en klasse for sig selv, *Utility*, som tilføjet forneden i DCDet med fed skrift (bemærk: navngivningskonvention i DCDet er som angivet i opgave Ex15-Disaheim):



Udfør følgende:

* Inden du begynder at ændre dit Disaheim-projekt, da **gem den nuværende version i GitLab**, så du altid kan vende tilbage til denne version (faktisk gør det for hver ny version af Disaheim, du laver fremover)
* **Implementér Utility-klassen**

## Øvelse 2.2: Test af Utility

Nu er det tid til en test. Udfør følgende:

* Opret en ny test-fil – *UnitTest2.cs* – i dit eksisterende test-projekt ”DisaheimTest”, dvs.:
  + I test-projektet tilføj en ny C#-klasse med navnet ”*UnitTest2.cs*”
  + Slet al kode i den nye fil, indsæt i stedet for:

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Disaheim;

namespace DisaheimTest

{

// Indsæt testklasse her!

}

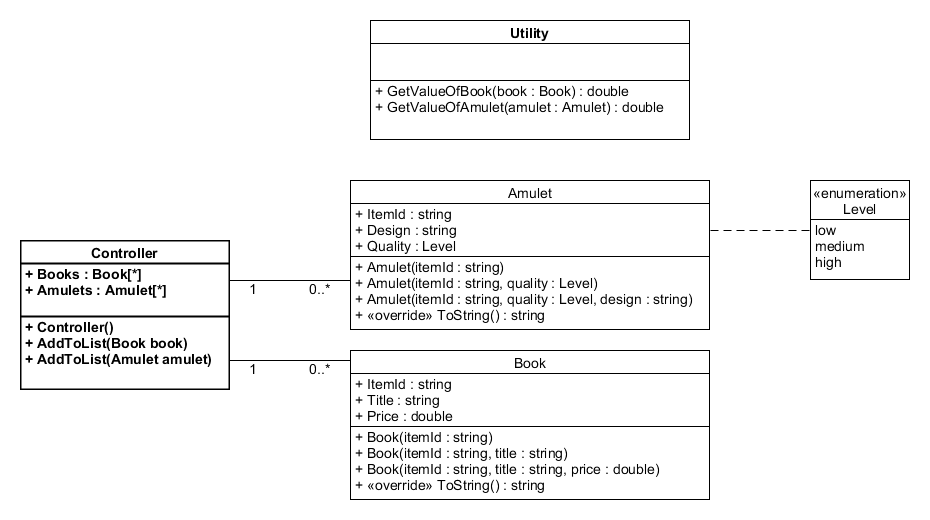
Så har du oprettet grundstrukturen i en ny testklasse.

* Indsæt **inde i** namespacet’s kodeblok indholdet af unit-testen ”*Ex16-Disaheim2.UnitTest2.docx*”, som hentes i git materiale-mappen
* Få testen til at køre succesfuldt. Ret din kode om nødvendigt

# Øvelse 3: Samlinger af bøger og amuletter

Disaheim ønsker selvfølgelig at have med mange bøger og amuletter at gøre. Der er derfor behov for at repræsentere samlinger (collections) af bøger og amuletter. Du tænker sikkert arrays, når du skal repræsentere en samling i C#, men nu skal du i stedet arbejde med List-datatypen, som på mange måder er nemmere at arbejde med end arrays (på bekostning af lidt hastighed).

Samlingerne af bøger og amuletter skal foreløbig håndteres af en anden klasse, nemlig Controller-klassen, som angivet i DCDet forneden til venstre:



## Øvelse 3.1: Implementér Controller-klassen

Udfør følgende:

* Implementér Controller-klassen i C#, hvor du anvender List<T>-datatypen i stedet for arrays til de to C#-properties Books og Amulets. Overvej hvad <T> står for, og hvordan det udtrykkes i din kode
* De to properties Books og Amulets skal initialiseres i constructor’en for Controller-klassen. Overvej hvad det indebærer. Hvad er nødvendigt for at initialisere de to properties?
* Implementér de to operationer AddToList() i designklassen som C# metoder. Kigger du nærmere efter, er der faktisk kun én operation, men med to forskellige typer af parametre. Dette kan løses med metode-overloading i C#

**Metode-overloading** er, som tidligere nævnt, en måde at implementere **polymorfi** på, den anden af de fire objekt-orienterede programmeringsprincipper, du indtil nu har set på (kan du huske den første, du lærte?).

## Øvelse 3.2: Test af Controller-klassen

Nu er det tid til at teste din implementering. Udfør følgende:

* Opret en ny test-fil – *UnitTest3.cs* – i dit test-projekt ”DisaheimTest”, dvs.:
  + I test-projektet tilføj en ny C#-klasse med navnet ”*UnitTest3.cs*”

Slet al kode i den nye fil, indsæt i stedet for:

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Disaheim;

namespace DisaheimTest

{

// Indsæt testklasse her!

}

Så har du oprettet grundstrukturen i en ny testklasse.

* Indsæt **inde i** namespacet’s kodeblok indholdet af unit-testen ”*Ex16-Disaheim2.UnitTest3.docx*” (git materiale-folderen)
* Få testen til at køre succesfuldt. Ret din kode om nødvendigt

## Øvelse 3.3: Debugging

Debug Controller-klassen i unit-testen:

* Indsæt et breakpoint (F9: Toggle Breakpoint) i Init()-metoden i UnitTest3.cs (dvs. i [TestInitialize]-sektionen), hvor controller-feltet instantieres: controller = new Controller();
* I Test Explorer vælg testmetoden, højreklik den, og vælg Debug-kommandoen
* Brug Step Into (F11) og Step Over (F10) til **at følge og tjekke din Controller-kode trin for trin** først i constructor’en i Controller og dernæst AddToList-metoderne, så du får en synlig forståelse for, hvad din kode egentlig gør