# Disaheim 5

|  |  |
| --- | --- |
| **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]** | Designsporet:   * [Unistrukturel] Du kan huske enkelte kendetegn ved **afhængighedshåndtering (en: dependency management)** – lagdeling (en: layering), pakkediagram (en: package diagram) * [Unistrukturel] Du kan genkende enkelte kendetegn ved **Designmønstre (en: design patterns)** – Repository pattern   Programmeringssporet:   * [Unistrukturel] Du kan huske enkelte kendetegn ved styring af **program flow** * [Unistrukturel] Du kan huske enkelte kendetegn ved **datahåndtering** * [Unistrukturel] Du kan genkende enkelte kendetegn ved simple **C# datatyper** * [Unistrukturel] Du kan genkende enkelte kendetegn ved **C# klasse** –  […], ’named’ og ’optional’ parameter i metode * [Unistrukturel] Du kan nævne enkelte kendetegn ved **C# interface** – naming, property, method * [Unistrukturel] Du kan nævne enkelte kendetegn ved **persistens** – File I/O, streams, tekstfil |
| **Din forberedelse** | Programmeringssporet:   * [Part 1: Getting Started with C# Interfaces](https://www.youtube.com/watch?v=jh4y8HzGqCQ) (video: 9:22) * [Interface Naming Guidelines](https://msdn.microsoft.com/en-us/library/8bc1fexb(v=vs.71).aspx) * [Part 2: Understanding C# Interfaces and Polymorphic Behavior](https://vimeo.com/254573540) (video: 7:11) * [Named and Optional Arguments](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/named-and-optional-arguments#optional-arguments) (læs de to afsnit og tilhørende eksempler) * Genopfrisk:   + [YB] 3.6 (s. 73-78) Using Files   + [How to read from and write to a text file](https://support.microsoft.com/en-us/help/816149/how-to-read-from-and-write-to-a-text-file-by-using-visual-c) * [While iterations and reading data from text file](https://channel9.msdn.com/Series/C-Fundamentals-for-Absolute-Beginners/12) (video: 14:24) * [File.ReadAllLines Method](https://docs.microsoft.com/en-us/dotnet/api/system.io.file.readalllines?view=netcore-3.1#code-try-1) * [Static Classes and Static Class Members](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/static-classes-and-static-class-members) |

I denne opgave introduceres du til C# interfaces. Sammen med klasser giver interfaces dig mulighed for at skabe og håndhæve fælles kontrakter på tværs af klasser, noget som .NET 6 og .NET Framework allerede gør i meget stor stil. Udover at en klasse kan ’nedarve’ fra mere end et interface (implementere et interface er en bedre formulering), så kan interfaces benyttes som datatyper for variable, noget man lige skal vænne sig til, men som udvider udtryksmulighederne med klasser endnu mere. Du kommer også til at genopfriske skrivning til og læsning fra en tekstfil, men denne gang defineret ud fra et interface.

# Dagens ord:

The critical design tool for software development  
 is a mind well educated in design principles.   
It is not UML or any other technology.  
(Craig Larman)

# Øvelse 1: Terminologi

Del teamet op i to mindre grupper, og brug **Ordet rundt** til at reflektere over begreberne ”interface”, herunder ”nedarvning af et interface”, ”implementering af interface i en klasse” og ”interface som variabel”. Sørg for at alle får mulighed for at tale.

*Tidsramme: 15 minutter*

**Benyt parprogrammering til alle følgende øvelser.**

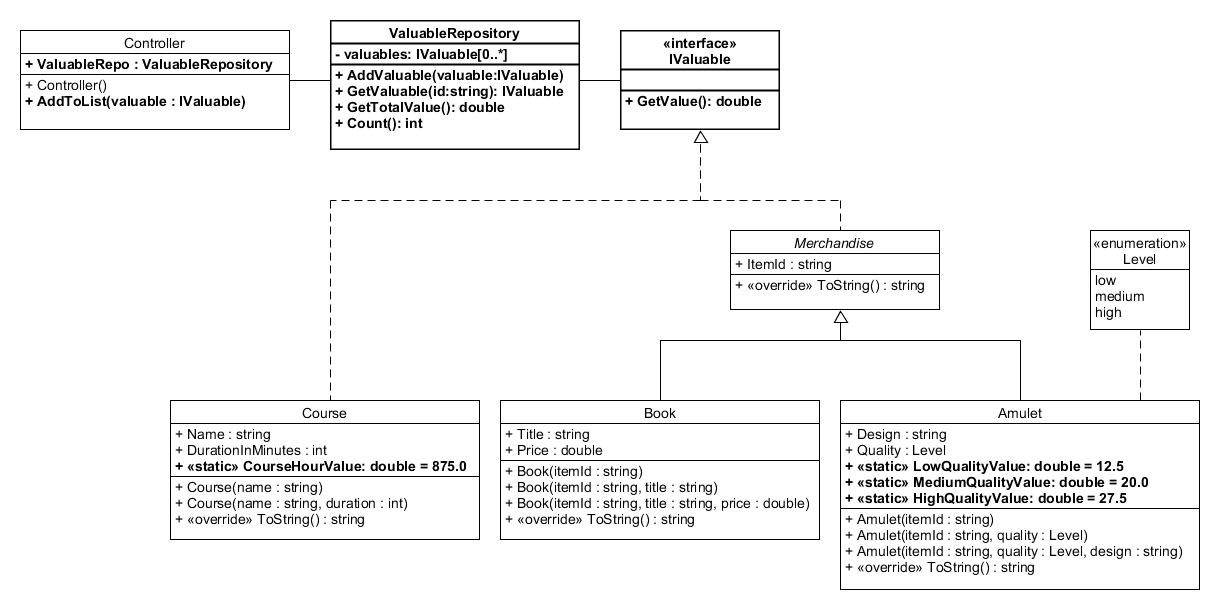
# Øvelse 2: Værdisættelse i Disaheim

Disaheim-designet er stadig ikke helt i orden. Virksomheden vil gerne have, at egenskaben ”at kunne værdi-sættes” følger objekterne (og dermed klasserne). Det er ikke nok at tilføje en metode i *Merchandise-klassen*, idet *Course* ikke tilhører dette arve-hierarki.

## Øvelse 2.1: Implementér med IValuable-interface

Et interface er velegnet til at beskrive egenskaber (kontrakter eller løfter), som skal findes på tværs af arve-hierarkier. Du skal derfor implementere et interface, *IValuable*, som definerer egenskaben at kunne værdi-sættes vha. metoden *GetValue()*. Det betyder, at klassen *Utility* (og dermed biblioteket) nu er helt overflødig, idet metoder og properties vedr. værdisætning nu flyttes til de respektive klasser. Vi kan nu også nøjes med ét repository – nemlig et som repræsenterer elementer af *IValuable*-typen.

Følgende diagram summerer alle disse ændringer (bemærk stereotypen «interface»):



Kig nøje på diagrammet, og udfør følgende:

* Implementér interfacet *IValuable* først. Som angivet i DCD’et, sørg for, at *Merchandise*-klassen og *Course*-klassen nedarver fra (dvs. implementerer) *IValuable*.
* Da *utility*-klassen er overflødig, flyttes *CourseHourValue*, *LowQualityValue*, *MediumQualityValue* og *HighQualityValue* til deres respektive klasser, som angivet i DCD’et. Bemærk, at disse 4 properties nu skal implementeres i C# som static properties. Overvej konsekvensen af denne ændring.
* Implementér *IValuable*-interfacets *GetValue*()-metode konkret i *Amulet*-, *Book*- og *Course*-klasserne. Hvad der skal stå i *Merchandise*-klassen for, at *GetValue*() kan implementeres i *Amulet*- og *Book*-klassen? (Hint: du kan med fordel benytte en abstrakt metode til dette)
* Implementér *ValuableRepository*-klassen. Bemærk, at metoden *Count()* i *ValuableRepository* returnerer antallet af elementer i repositoriet.

## Øvelse 2.2: Test af interface-implementering

Udfør følgende:

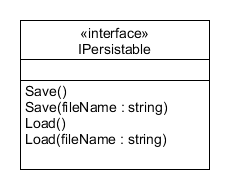
* Opret en ny test-fil – *UnitTest5.cs* – i dit test-projekt ”DisaheimTest” som tidligere vist.
* Indsæt indholdet af unit-testen ”*Ex19-Disaheim5.UnitTest5.docx*”, der ligger i git materiale-mappen.
* Få testen til at køre succesfuldt. Ret din kode om nødvendigt.

# Øvelse 3: Gemme og hente økonomioplysninger på harddisken

Disaheim vil gerne have mulighed for at gemme og hente sine oplysninger på harddisken. Denne funktionalitet er temmelig generel og kan bruges i mange forskellige klasser, så du beslutter dig for at definere en kontrakt (interface) med de relevante metoder, inden du implementerer denne funktionalitet.

## Øvelse 3.1: IPersistable interface

DesignAway-firmaet har leveret følgende design af interfacet:



Udfør følgende:

* Definér interfacet *IPersistable* i C# i din Disaheim-løsning ud fra ovenstående model.

## Øvelse 3.2: Implementér IPersistable for ValuableRepository

Du skal nu implementere IPersistable-interfacet for din ValuableRepository-klasse. Til dette formål har designerne leveret følgende operationskontrakter for IPersistable specifik for ValuableRepository:

Operationskontrakt for **Save()**

Cross reference: Budgetopfølgning

Precondition: Ingen

Postcondition: Ingen ændringer i systemet

Output: Der er oprettet en fil med navnet ”ValuableRepository.txt”. Filen indeholder en repræsentation af alle de objekter *ValuableRepository* opbevarer, en linje for hvert objekt. I en linje står først objektets mest specifikke type og dernæst værdierne af alle properties, adskilt med et semikolon (’;’).   
Eksempel på en bog: ”BOG;Id3;Spirits in the Night;123.55”  
Hvis filen eksisterer, overskrives den uden advarsel.

Operationskontrakt for **Save(fileName : string)**

Cross reference: Budgetopfølgning

Precondition: Ingen

Postcondition: Ingen ændringer i systemet

Output: Som Save(), bare med angivne filnavn.

Operationskontrakt for **Load()**

Cross reference: Budgetopfølgning

Precondition: Ingen

Postcondition: Filen ” ValuableRepository.txt” indlæses og for hver linjen i filen instantieres et objekt, som tilføjes *ValuableRepository*. Den semikolon-separerede tekstlinje leverer værdier til objektets properties (første værdi angiver objektets mest specifikke type, følgende værdier angiver properties).  
Hvis filen ikke eksisterer, er indholdet af repository tomt.

Operationskontrakt for **Load(fileName : string)**

Cross reference: Budgetopfølgning

Precondition: Ingen

Postcondition: Som Load(), bare med angivne filnavn.

Udfør følgende:

* Lad ValuableRepository nedarve fra IPersistable
* Implementér interfacet i ValuableRepository.

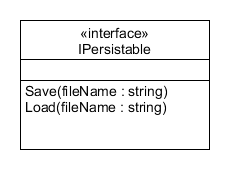
## Øvelse 3.3: Test interface implementeringen

Udfør følgende:

* Opret en ny test-fil – *UnitTest6.cs* – i dit test-projekt ”DisaheimTest” som tidligere vist.
* Indsæt indholdet af unit-testen ”*Ex19-Disaheim5.UnitTest6.docx*”, der ligger i git materiale-mappen.
* Få testen til at køre succesfuldt. Ret din kode om nødvendigt.

## Øvelse 3.4: Valgfri parametre (en : optional arguments)

Designerne vil efter lidt overvejelse gerne gøre brug af valgfrie parametre i IPersistable-interfacet. De har derfor ændret det lidt:



Operationskontrakterne foroven har angivet et default-filnavn, hvor økonomioplysninger gemmes i.

Udfør følgende:

* Tilret IPersistable-interfacet jf. ovenstående.
* Tilret implementering af det ændrede interface i ValuableRepository-klassen
* Kør UnitTest6 igen. Den skulle gøre køre fejlfrit uden ændringer. Tilret i din kode (ikke testen), hvis nødvendigt.