# Custom Exception Handling

|  |  |
| --- | --- |
| **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 genkende enkelte kendetegn ved **strukturmodellering (en: modeling structure)** – designklassediagram (DCD) * [Unistrukturel] Du kan genkende enkelte kendetegn ved **adfærdsmodellering (en: modeling behaviour)** – sekvensdiagram (SD)   Programmeringssporet:   * [Unistrukturel] Du kan huske enkelte kendetegn ved styring af **program flow** – throw, try-catch, try-finally, try-catch-finally * [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 (objekt)** |
| **Din forberedelse** | Programmeringssporet:   * [Common exceptions](https://docs.microsoft.com/en-us/dotnet/standard/exceptions/#common-exceptions) * [Exception class and properties](https://docs.microsoft.com/en-us/dotnet/standard/exceptions/exception-class-and-properties) * [Exception handling](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/exceptions/exception-handling) * [throw](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/throw), [try-catch](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/try-catch), [try-finally](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/try-finally) og [try-catch-finally](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/try-catch-finally) * [Gracefully Handling Exceptions](https://channel9.msdn.com/Series/CSharp-Fundamentals-for-Absolute-Beginners/Gracefully-Handling-Exceptions) (video: 21:13)   + Genlæs evt. [File ReadAllLines](https://docs.microsoft.com/en-us/dotnet/api/system.io.file.readalllines?view=netcore-3.1#code-try-1) først! * [Best practices for exceptions](https://docs.microsoft.com/en-us/dotnet/standard/exceptions/best-practices-for-exceptions#end-exception-class-names-with-the-word-exception) * [How to create user-defined exceptions](https://docs.microsoft.com/en-us/dotnet/standard/exceptions/how-to-create-user-defined-exceptions) * Se på de udleverede Tusindfryd analyse-artefakterne *igen* (OM, DM, UC, SSD, OC) med fokus på systemsekvensdiagrammet ”Etabler ny blomstersort” |

I dagens opgave skal du arbejde lidt dybere med exceptions og bl.a. definere dine egne brugerdefinerede exception. Derudover skal du arbejde videre med Tusindfryd-casen.

# Dagens citat:

UML is not dessert topping and floor wax.  
([Grady Booch](https://en.wikipedia.org/wiki/Grady_Booch))

# Øvelse 1: Terminologi

Del teamet op i to mindre grupper, og brug **Møde på midten** i hver gruppe til at diskutere begreberne ”Exception”, ”throw”, ”try-catch”, ”try-finally” og ”try-catch-finally”. Husk, at alle i gruppen skal have taletid.

# Øvelse 2: Exception

Du skal nu se på håndtering af fejl i din kode med brug af nogle af de indbyggede Exception-klasser i .Net samt definere egne exceptions. Benyt parprogrammering til denne øvelse.

## Øvelse 2.1: Klargøring

Udfør følgende:

* Opret en konsolapplikation med navnet ”*CustomExceptionHandling*”.
* Tilføj et nyt bibliotek (Class Library) til din løsning med navnet ”*TotallySafeLib*”, opret her i en ny C#-klasse *TotallySafe*, og indsæt her i koden fra”*Ex21-CustomExceptionHandling.TotallySafeLib.docx*”, der ligger git materiale-mappen.
* Sørg for, at din konsolapplikation kender til dit nye bibliotek.

## Øvelse 2.2: Inspektion og afprøvning

I denne øvelse skal du inspicere implementeringen af metoden *GetValueAtPosition* i dit nye bibliotek.

Overvej først følgende:

* Hvad vil blive returneret, hvis du benyttede *GetValueAtPosition*-metoden med heltallet ”1” som input-argument?
* Hvad vil der ske, hvis du benyttede dig af heltallet ”4” som input?
* Hvad vil der ske, hvis du benyttede dig af heltallet ”9” som input?
* Hvad vil der ske, hvis du benyttede dig af heltallet ”-2” som input?

Indsæt følgende kode i din Main-metode:

static void Main(string[] args)

{

int posValue = TotallySafe.GetValueAtPosition(<indsæt heltal>);

Console.WriteLine(posValue);

Console.ReadLine();

}

* Overvej, hvorfor du ikke behøver at instantiere TotallySafe-klassen, før du bruger den.

Indsæt herefter (*en af gangen*) følgende heltalsargument i stedet for <indsæt heltal>:

* Start med heltalsargumentet 1
* Benyt herefter heltallet 4
* Benyt derefter heltallet 9
* Benyt endelig heltallet -2

Var dine antagelser korrekte?

## Øvelse 2.3: Håndtering af exception

Det, du oplevede i øvelse 2.2, var en *IndexOutOfRangeException*. Du fik desuden at vide, at du ikke havde håndteret din exception (en: ’Exception Unhandled’). Denne håndtering skal du se nærmere på nu.

Udfør følgende:

* Benyt dig af en try-catch-konstruktion i din Main-metode til at håndtere den problematiske kode (fra øvelse 2.2)
* Udskriv den generede Exception-besked ud til skærmen (hint: *ex.Message*)
* Benyt dig af den specifikke Exception-klasse frem for base-klassen (hint: [Common exceptions](https://docs.microsoft.com/en-us/dotnet/standard/exceptions/#common-exceptions))

## Øvelse 2.4: Inspektion og afprøvning

Der er to metoder mere inde i dit bibliotek, *Divider* og *StringToInt*. Overvej om der kunne være et problem med deres implementering ud fra en visuel inspektion. Benyt dem herefter i din konsolapplikation og afprøv din formodning. Endeligt skal du håndtere disse fejl med den specifikke exception-klasse (igen frem for base-klassen).

## Øvelse 2.5: Brugerdefineret exception

Du vil nu gerne ændre en af metoderne i TotallySafe-biblioteket, så den eksplicit kaster en exception. Du er dog ikke helt tilfreds med at bruge en af de indbyggede Exception-klasser, så du skal implementere din helt egen og bruge den.

Hvis du kalder *GetValueAtPosition*-metoden med et negativt argument, f.eks. -2, så kastes der automatisk en *IndexOutOfRangeException*. Det vil du gerne forbedre, så du benytter din helt egen exception til at håndtere et negativt argument.

Udfør følgende trin:

* Definér din egen brugerdefinerede exception-klasse *NegativeIndexOutOfRangeException*, dvs. opret en ny klasse til dette formål i dit TotallySafe-bibliotek. Sørg for, at din nye exception-klasse implementerer de tre constructors nævnt i forberedelsen [How to create user-defined exceptions](https://docs.microsoft.com/en-us/dotnet/standard/exceptions/how-to-create-user-defined-exceptions).
* Opdatér koden i *GetValueAtPosition*-metoden, så den kaster din nye exception med en passende exception-tekst, hvis indekset angivet i *positionInArray*-parameteren er negativt.
* Opdatér koden i *Main*-metoden, så den også fanger din nye exception (samtidig med den automatisk genererede).
* Afprøv med et kald til *GetValueAtPosition*-metoden med et negativt argument og se, om din nye exception faktisk virker.

# Øvelse 3: Tusindfryd

Du og dit team har i forrige opgave Ex20 udarbejdet en DCD og enkelte SD’er for Tusindfryd-casen. Du skal arbejde videre med disse artefakter, hvor du skal udforme et SD ud fra en OC – og derefter kode ud fra jeres SD. Det er vigtigt, at du først koder efter, du har udarbejdet din SD. Benyt parprogrammering til øvelsen.

## Øvelse 3.1: SD ud fra OC

Du og dit team skal nu udforme et sekvensdiagram (SD) samt tilpasse DCD’et ud fra operationen *etablerNyBlomstersort*; hertil har du fået udleveret følgende operationskontrakt (OC):

|  |  |
| --- | --- |
| **Operation:** | etablerNyBlomstersort(navn:string, billede:binary, produktionstid:integer, halveringstid:integer, størrelse:integer) |
| **Cross References:** | Etabler ny blomstersort |
| **Preconditions:** |  |
| **Postconditions:** | * et blomsterobjekt med navn, billede, produktionstid, halveringstid og størrelse blev oprettet (instance creation) |

Sørg for, at dine kvalitetskriterier for SD og DCD bliver overholdt.

## Øvelse 3.2: Kvalitetssikring af SD

Når du og dit team er færdige med at udforme jeres SD, skal I have kvalitetssikret jeres artefakt med minimum et andet team.

Lav rettelser til jeres artefakt og kvalitetskriterier ud fra behov.

## Øvelse 3.3: Implementering af kode ud fra SD

Med udgangspunkt i den udarbejdede SD og tilhørende DCD, skal du nu implementere din SD i kode. Dvs. opret en passende konsolapplikation i Visual Studio, hvor du implementerer de relevante klasser i C# samt den udvalgte SD. Det er tilladt at oprette ’fake data’.

**Bemærk:** Du og din sidemakker må gerne spørge dine medstuderende/os undervisere/lektiecafé-hjælperne til råds, hvis I er gået i stå.