# Objekt-Orienteret Programmering

## Exceptions

Aslak Johansen asjo@mmmi.sdu.dk
Peter Nellemann pmn@mmmi.sdu.dk

September 25, 2024

SDU

# Part 0:
# Robusthed

# Robusthed

Et *robust* program er et, der opfører sig på en hensigtsmæssig måde, når der sker noget uventet. Vi siger da at programmet er robust overfor denne specifikke klasse af hændelser.

# Robusthed

Et *robust* program er et, der opfører sig på en hensigtsmæssig måde, når der sker noget uventet. Vi siger da at programmet er robust overfor denne specifikke klasse af hændelser.

Eksempler på sådanne klasser af hændelser:

- Mistet netværksforbindelse
- Fejl ved læsning af fil
- Uventet brugerinput

# Robusthed

Et *robust* program er et, der opfører sig på en hensigtsmæssig måde, når der sker noget uventet. Vi siger da at programmet er robust overfor denne specifikke klasse af hændelser.

Eksempler på sådanne klasser af hændelser:

- Mistet netværksforbindelse
- Fejl ved læsning af fil
- Uventet brugerinput

Vi kan opnå robusthed ved at teste for alt, eksempelvis ved brug af branches. Dette er dog:

- Meget kode
- Meget kompleksitet
- Meget arbejde

# Robusthed

Et *robust* program er et, der opfører sig på en hensigtsmæssig måde, når der sker noget uventet. Vi siger da at programmet er robust overfor denne specifikke klasse af hændelser.

Eksempler på sådanne klasser af hændelser:

- Mistet netværksforbindelse
- Fejl ved læsning af fil
- Uventet brugerinput

Vi kan opnå robusthed ved at teste for alt, eksempelvis ved brug af branches. Dette er dog:

- Meget kode
- Meget kompleksitet
- Meget arbejde

$\Rightarrow$ Flere fejl

# Robusthed

Et *robust* program er et, der opfører sig på en hensigtsmæssig måde, når der sker noget uventet. Vi siger da at programmet er robust overfor denne specifikke klasse af hændelser.

Eksempler på sådanne klasser af hændelser:

- Mistet netværksforbindelse
- Fejl ved læsning af fil
- Uventet brugerinput

Vi kan opnå robusthed ved at teste for alt, eksempelvis ved brug af branches. Dette er dog:

- Meget kode
- Meget kompleksitet
- Meget arbejde

$\Rightarrow$ Flere fejl

I C# bruger man ofte *exceptions* til at opnå robusthed ved minimal forøgelse af linjetallet.

# Part 1:
# Exceptions

Håndteringen af den uventede situationer i C# sker i to trin:

Håndteringen af den uventede situationer i C# sker i to trin:

1. ***Detektion*** Når det bliver klart at man står i en situation der ligger udenfor hvad den aktuelle implementation er designet til at kunne håndtere udføres følgende operationer:

# Exceptions ▷ Exception Koncept

Håndteringen af den uventede situationer i C# sker i to trin:

1. ***Detektion*** Når det bliver klart at man står i en situation der ligger udenfor hvad den aktuelle implementation er designet til at kunne håndtere udføres følgende operationer:
   1.1 En *exception* oprettes.

# Exceptions ▷ Exception Koncept

Håndteringen af den uventede situationer i C# sker i to trin:

1. ***Detektion*** Når det bliver klart at man står i en situation der ligger udenfor hvad den aktuelle implementation er designet til at kunne håndtere udføres følgende operationer:
   1.1 En *exception* oprettes.
   1.2 Denne exception *kastes*.

# Exceptions ▷ Exception Koncept

Håndteringen af den uventede situationer i C# sker i to trin:

1. ***Detektion*** Når det bliver klart at man står i en situation der ligger udenfor hvad den aktuelle implementation er designet til at kunne håndtere udføres følgende operationer:
   1.1 En *exception* oprettes.
   1.2 Denne exception *kastes*.
2. ***Håndtering*** Den kastede exception *gribes* og dette trigger udførelsen af en block.

# Exceptions ▷ Exception Koncept

Håndteringen af den uventede situationer i C# sker i to trin:

1. ***Detektion*** Når det bliver klart at man står i en situation der ligger udenfor hvad den aktuelle implementation er designet til at kunne håndtere udføres følgende operationer:
   1.1 En *exception* oprettes.
   1.2 Denne exception *kastes*.
2. ***Håndtering*** Den kastede exception *gribes* og dette trigger udførelsen af en block.

Disse to trin er adskilt således at detektion og håndtering kan implementeres i to vidt forskellige funktioner.

Håndteringen af den uventede situationer i C# sker i to trin:

1. ***Detektion*** Når det bliver klart at man står i en situation der ligger udenfor hvad den aktuelle implementation er designet til at kunne håndtere udføres følgende operationer:
   1.1 En *exception* oprettes.
   1.2 Denne exception *kastes*.
2. ***Håndtering*** Den kastede exception *gribes* og dette trigger udførelsen af en block.

Disse to trin er adskilt således at detektion og håndtering kan implementeres i to vidt forskellige funktioner.

Kastet har en retning, og denne retning definerer hvor exceptionen kan gribes og dermed behandles.

# Exceptions ▷ Klassen `Exception`

Exceptions benyttes til at indikere at noget afviger fra den forventede opførsel i programmet.

# Exceptions ▷ Klassen `Exception`

Exceptions benyttes til at indikere at noget afviger fra den forventede opførsel i programmet.

Når exceptions kastes på grund af eksterne omstændigheder, bør vi typisk håndtere det med exception handling. Fx når:

- Vi mister netværksforbindelsen.
- Vi prøver at tilgå en fil der ikke eksisterer.

# Exceptions ▷ Klassen `Exception`

Exceptions benyttes til at indikere at noget afviger fra den forventede opførsel i programmet.

Når exceptions kastes på grund af eksterne omstændigheder, bør vi typisk håndtere det med exception handling. Fx når:

- Vi mister netværksforbindelsen.
- Vi prøver at tilgå en fil der ikke eksisterer.

Men når de kastes som et resultat af en programmeringsfejl bør vi generelt udelade exception handling og i stedet lade programmet crashe således at fejlen kan rettes. Fx når:

- Vi indekserer ind i et array udenfor den allokerede størrelse.
- Vi følger en `null` reference.

# Part 2:
# Exception Handling

# Exception Handling ▷ Introduktion

Eksempel:

```
void work (int count) {
  Console.WriteLine("A slice has to be "+(360/count)+" degrees");
}

string[] children = new string[] {"Aslak", "Peter"};
work(children.Length);
```

# Exception Handling ▷ Introduktion

Eksempel:

```
void work (int count) {
  Console.WriteLine("A slice has to be "+(360/count)+" degrees");
}

string[] children = new string[] {"Aslak", "Peter"};
work(children.Length);
```

Lad os udføre programmet:

```
aslak@gaia:/tmp/cake$ dotnet run
A slice has to be 180 degrees
aslak@gaia:/tmp/cake$
```

# Exception Handling ▷ Introduktion

Eksempel:

```
void work (int count) {
  Console.WriteLine("A slice has to be "+(360/count)+" degrees");
}

string[] children = new string[] {};
work(children.Length);
```

# Exception Handling ▷ Introduktion

Eksempel:

```
void work (int count) {
  Console.WriteLine("A slice has to be "+(360/count)+" degrees");
}

string[] children = new string[] {};
work(children.Length);
```

Lad os udføre programmet:

```
aslak@gaia:/tmp/cake$ dotnet run
Unhandled exception. System.DivideByZeroException: Attempted to divide by zero.
   at Program.<<Main>$>g__work|0_0(Int32 count) in /tmp/cake/Program.cs:line 2
   at Program.<Main>$(String[] args) in /tmp/cake/Program.cs:line 6
aslak@gaia:/tmp/cake$
```

# Exception Handling ▷ Den Ugrebne Exception

Lad os se lidt mere på hvad der sker når en exception ikke gribes (eng: is uncaught):

```
aslak@gaia:/tmp/cake$ dotnet run
Unhandled exception. System.DivideByZeroException: Attempted to divide by zero.
   at Program.<<Main>$>g__work|0_0(Int32 count) in /tmp/cake/Program.cs:line 2
   at Program.<Main>$(String[] args) in /tmp/cake/Program.cs:line 6
```

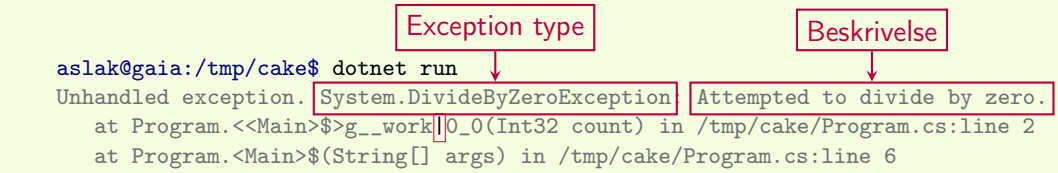
# Exception Handling ▷ Den Ugrebne Exception

Lad os se lidt mere på hvad der sker når en exception ikke gribes (eng: is uncaught):


```
aslak@gaia:/tmp/cake$ dotnet run
Unhandled exception. System.DivideByZeroException: Attempted to divide by zero.
   at Program.<<Main>$>g__work|0_0(Int32 count) in /tmp/cake/Program.cs:line 2
   at Program.<Main>$(String[] args) in /tmp/cake/Program.cs:line 6
```

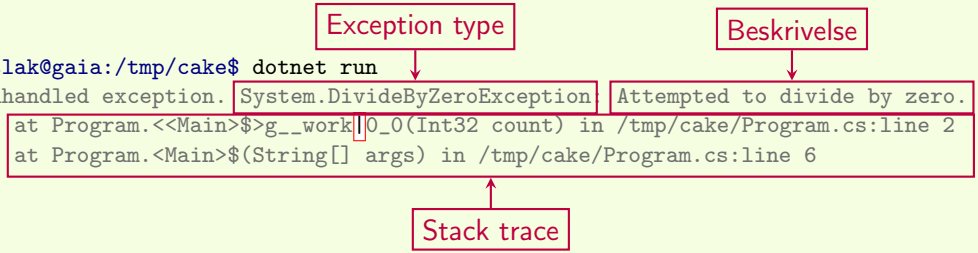
# Exception Handling ▷ Den Ugrebne Exception

Lad os se lidt mere på hvad der sker når en exception ikke gribes (eng: is uncaught):



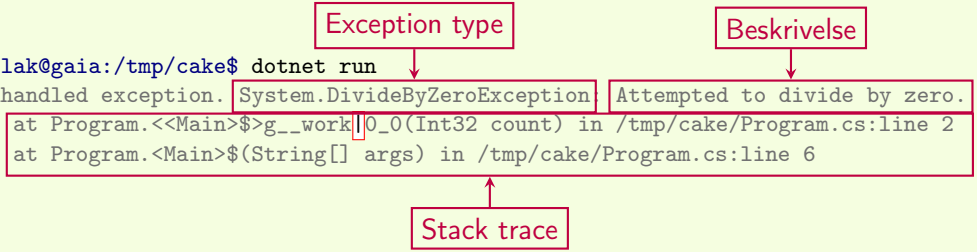
# Exception Handling ▷ Den Ugrebne Exception

Lad os se lidt mere på hvad der sker når en exception ikke gribes (eng: is uncaught):



# Exception Handling ▷ Den Ugrebne Exception

Lad os se lidt mere på hvad der sker når en exception ikke gribes (eng: is uncaught):




Når en exception ikke gribes af vores kode afsluttes vores program ved at evaluere til denne exception og CLR (C#'s virtuelle maskine) præsenterer os for de data der er indeholdt i den.

# Exception Handling ▷ Håndtering Indenfor Stakramme

```
void work (int count) {
  try {
    Console.WriteLine("A slice has to be "+(360/count)+" degrees");
  } catch (DivideByZeroException) {
    Console.WriteLine("No-one to eat the cake. Sad!");
  }
}

string[] children = new string[] {};
work(children.Length);
```

# Exception Handling ▷ Håndtering Indenfor Stakramme

```
void work (int count) {
  try {
    Console.WriteLine("A slice has to be "+(360/count)+" degrees");
  } catch (DivideByZeroException) {
    Console.WriteLine("No-one to eat the cake. Sad!");
  }
}

string[] children = new string[] {};
work(children.Length);

aslak@gaia:/tmp/cake$ dotnet run
No-one to eat the cake. Sad!
aslak@gaia:/tmp/cake$
```

# Exception Handling ▷ Håndtering via Stakken

```
void work (int count) {
  Console.WriteLine("A slice has to be "+(360/count)+" degrees");
}

string[] children = new string[] {};
try {
  work(children.Length);
} catch (DivideByZeroException) {
  Console.WriteLine("No-one to eat the cake. Sad!");
}
```

# Exception Handling ▷ Håndtering via Stakken

```
void work (int count) {
  Console.WriteLine("A slice has to be "+(360/count)+" degrees");
}

string[] children = new string[] {};
try {
  work(children.Length);
} catch (DivideByZeroException) {
  Console.WriteLine("No-one to eat the cake. Sad!");
}
```

```
aslak@gaia:/tmp/cake$ dotnet run
No-one to eat the cake. Sad!
aslak@gaia:/tmp/cake$
```

# Exception Handling ▷ `try-catch` Konstruktionen

Syntaks:
$try\ \langle try - block \rangle\ \langle catch - list \rangle$
Hvor en $\langle catch - list \rangle$ er en sekvens bestående af én eller flere forekomster af én af følgende:
$catch\ (\langle type \rangle\ \langle variable - name \rangle)\ \langle catch - block \rangle$
$catch\ (\langle type \rangle)\ \langle catch - block \rangle$

# Exception Handling ▷ `try-catch` Konstruktionen

Syntaks:
*try* ⟨*try* − *block*⟩ ⟨*catch* − *list*⟩
Hvor en ⟨*catch* − *list*⟩ er en sekvens bestående af én eller flere forekomster af én af følgende:
*catch* (⟨*type*⟩ ⟨*variable* − *name*⟩) ⟨*catch* − *block*⟩
*catch* (⟨*type*⟩) ⟨*catch* − *block*⟩

Her er ⟨*type*⟩ en form for specifik exception type.

# Exception Handling ▷ `try-catch` Konstruktionen

Syntaks:
*try* ⟨*try* − *block*⟩ ⟨*catch* − *list*⟩
Hvor en ⟨*catch* − *list*⟩ er en sekvens bestående af én eller flere forekomster af én af følgende:
*catch* (⟨*type*⟩ ⟨*variable* − *name*⟩) ⟨*catch* − *block*⟩
*catch* (⟨*type*⟩) ⟨*catch* − *block*⟩

Her er ⟨*type*⟩ en form for specifik exception type.

Med konstruktionen kan man angive hvad der skal gribes (og dermed også hvad der skal passere), og hvad der skal gøres når sådanne exceptions gribes.

# Exception Handling ▷ Multiple `catch` Konstruktioner

```
try {
  Operation(1, 0);
}
catch (DivideByZeroException e) {
  Console.WriteLine("Unable to do division.");
  Console.WriteLine(e);
}
catch (NegativeDivisorException ex) {
  Console.WriteLine("Unable to do division.");
  Console.WriteLine(e);
}
```

# Exception Handling ▷ Multiple `catch` Konstruktioner

```
try {
  Operation(1, 0);
}
catch (DivideByZeroException e) {
  Console.WriteLine("Unable to do division.");
  Console.WriteLine(e);
}
catch (NegativeDivisorException ex) {
  Console.WriteLine("Unable to do division.");
  Console.WriteLine(e);
}
```

**Bemærk:** Catch-mønstre evalueres i rækkefølge, og kun det er kun blocken for det første match der udføres.

# Part 3:
# At Kaste med Exceptions

# Part 3:
# At Kaste med Exceptions
# (Når Man Selv Bor i et Glashus)

# Kast med Exceptions

Vi kan lave et `RuntimeException` objekt (der er unchecked) og *kaste* (eng: throw) det:

```
double divide (double a, double b)
{
  if (b == 0) {
    throw new Exception("B is equal to 0");
  }

  return a/b;
}
```

# Part 4:
# Propagering af Exceptions

# Propagering af Exceptions ▷ Regler

Når en exception kastes sker dette et bestemt **sted** i den fulde kodebase; enten i vores egen kode eller i noget kode andre har skrevet (men som vi benytter os af).

# Propagering af Exceptions ▷ Regler

Når en exception kastes sker dette et bestemt **sted** i den fulde kodebase; enten i vores egen kode eller i noget kode andre har skrevet (men som vi benytter os af).

C#'s fortolker afbryder da normal afvikling af programmet og – med udgangspunkt i dette sted – søger den efter en kompatibel exception handler.

# Propagering af Exceptions ▷ Regler

Når en exception kastes sker dette et bestemt **sted** i den fulde kodebase; enten i vores egen kode eller i noget kode andre har skrevet (men som vi benytter os af).

C#'s fortolker afbryder da normal afvikling af programmet og – med udgangspunkt i dette sted – søger den efter en kompatibel exception handler.

Denne søgning følger en helt bestemt algoritme:

1. Gennemsøg aktuelle metode med udgangspunkt i aktuelle position.
   - 1.1 Hvis den aktuelle block enten ikke er en `try`-block eller denne ikke har en matchende `catch`-block hoppes der til punkt 1.3.
   - 1.2 Den matchende `catch`-block udføres og derefter fortsætter programmets afvikling efter den aktuelle `try-catch` konstruktion.
   - 1.3 Hvis der er en ydre block så trædes der ud i denne og der hoppes til punkt 1.1.
2. Hvis vi er i `main` metoden konkluderer CLR at der ikke er nogen handler og afslutter programmet med en relevant fejlmeddelelse.
3. Den øverste stakramme (en: stack frame) poppes af og den tilstand som denne repræsenterer reetableres.
4. Hop til punkt 1.

# Propagering af Exceptions ▷ Genkast (eng: rethrowing)

```
int work (int dividend, int divisor) {
  return dividend/divisor;
}

int intermediary (int dividend, int divisor) {
  try {
    return work(dividend, divisor);
  } catch (DivideByZeroException e) {
    Console.WriteLine("Oh, shit!");
    throw new Exception("Divisor was invalid.", e);
  }
}

int[] divisors = new int[] {2, 1, 0, -1, -2};
foreach (int divisor in divisors) {
  Console.WriteLine(intermediary(42, divisor));
}
```

# Part 5:
# Finally

# Finally ▷ Syntaks

`finally` konstruktioner bruges til at erklære at noget kode skal afvikles uanset om vejen igennem en `try-catch` konstruktion går igennem `catch` clausen.

# Finally ▷ Syntaks

`finally` konstruktioner bruges til at erklære at noget kode skal afvikles uanset om vejen igennem en `try-catch` konstruktion går igennem `catch` clausen.

Syntaks:
*try* ⟨*try* − *block*⟩
⟨*optional* − *catch* − *clauses*⟩
⟨*finally* − *clause*⟩

# Finally ▷ Syntaks

`finally` konstruktioner bruges til at erklære at noget kode skal afvikles uanset om vejen igennem en `try-catch` konstruktion går igennem `catch` clausen.

Syntaks:
*try* ⟨*try* − *block*⟩
⟨*optional* − *catch* − *clauses*⟩
⟨*finally* − *clause*⟩

`finally` blocken vil blive udført efter `try` blocken (hvis der ikke bliver kastet en exception heri) eller efter den matchende `catch` block (hvis der bliver kastet en exception).

# Finally ▷ Syntaks

`finally` konstruktioner bruges til at erklære at noget kode skal afvikles uanset om vejen igennem en `try-catch` konstruktion går igennem `catch` clausen.

Syntaks:
*try* ⟨*try* − *block*⟩
⟨*optional* − *catch* − *clauses*⟩
⟨*finally* − *clause*⟩

`finally` blocken vil blive udført efter `try` blocken (hvis der ikke bliver kastet en exception heri) eller efter den matchende `catch` block (hvis der bliver kastet en exception).

**Bemærk:**

- Hvis der i `try` blocken bliver kastet en exception som ikke gribes i en `catch` block så vil `finally` blocken *ikke* blive afviklet.
- Hverken `catch` eller `finally` er obligatoriske.
- Mindst én konstruktion af `catch` og `finally` typen skal indgå.

# Finally ▷ Eksempel

`finally` konstruktioner bruges ofte til at håndtere problemer der opstår udenfor vores program.

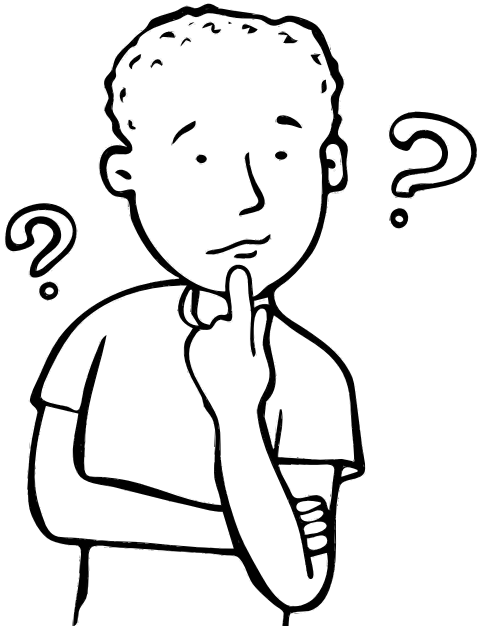
# Finally ▷ Eksempel

`finally` konstruktioner bruges ofte til at håndtere problemer der opstår udenfor vores program.

```
try {
  // open a network connection
  // communicate over the connection
}
catch (IOException e) {
  // report the error
}
finally {
  // close the connection
}
```


I CAN'T HANDLE THAT EXCEPTION
YOU DON'T NEED TO
imgflip.com

# Questions?



https://openclipart.org/detail/236687/boy-thinking-of-question