Exceptions

OOP 2020

Thomas Bøgholm
boegholm@cs.aau.dk

Opfølgning

Miniprojekt

Fejlhåndtering i C#

- Hvordan gør man i C?

Fejlhåndtering del 1: **håndtering**

- Exception-håndtering er delt i to: først **Håndtering**
  - try
  - catch (et antal)
  - finally

Fejlhåndtering

- Exce
  - try
  - ca
  - fin

Ide:

Prøv noget (try) og lad som om det går godt.

Fejlhåndtering

- Exce
  - try
  - ca
  - fin

Ide:

Prøv noget (try) og lad som om det går godt.

Hvis det så ikke går godt, afbryd, og hop hen et andet sted hvor du håndterer fejlen

Eksempler på fejlsituationer

```
void Foo()
{
    Person p = null;
    p.FirstName = "Thomas"; // fejl!
    p.LastName = "Bøgholm"; // fejl!
    p.Print();              // fejl!
}
```

Eksempler på fejlsituationer

```
void Foo()
{
    Person p = null;
    p.FirstName = "Thomas"; // fejl!
    p.LastName = "Bøgholm"; // fejl!
    p.Print();              // fejl!
}
```

```
void Bar()
{
    Person p = new Person()
    p.FirstName = "Thomas";
    p.LastName = "Bøgholm";
    Elefant e = (Elefant)p; // fejl!
}
```

Håndtering af nullreference-fejl

```
void Foo()
{
    Person p = null;
    p.FirstName = "Thomas"; // fejl!
    p.LastName = "Bøgholm"; // fejl!
    p.Print();              // fejl!
}
```

Håndtering af nullreference-fejl

```
try {
    Person p = null;
    p.FirstName = "Thomas"; // fejl! – p er null!
    p.LastName = "Bøgholm"; // fejl! – p er stadig null
    p.Print();              // fejl! - ..
} catch (NullReferenceException e)
{
    Console.WriteLine("Der skete null-reference fejl!");
    Console.WriteLine(e.Message);
}
```

Cast-fejl

```
void Bar()
{
    Person p = new Person();
    p.FirstName = "Thomas";
    p.LastName = "Bøgholm";
    Elefant e = (Elefant)p; // fejl!
}
```

Cast-fejl

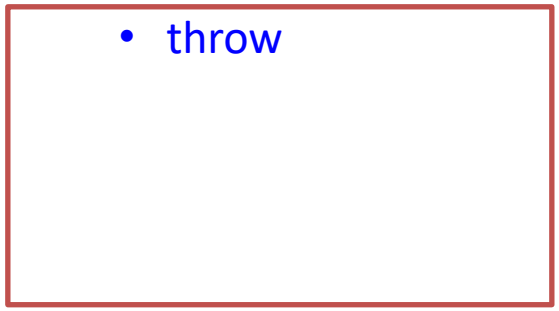
```
void Bar()
{
    Person p;
    Elefant e;
    try {
        p = new Person();
        p.FirstName = "Thomas";
        p.LastName = "Bøgholm";
        e = (Elefant)p; // fejl!
    } catch (InvalidCastException e)
    {
        Console.WriteLine("Fejl i cast!");
    }
}
```

```
static void Main(string[] args)
{
    try
    {
        //kan give FormatException
        int a = int.Parse(Console.ReadLine());
        int b = int.Parse(Console.ReadLine());

        //kan give DivideByZeroException
        double c = (double)a / b;
    }
    catch (FormatException formatError)
    {
        Console.WriteLine(formatError.Message);
    }
    catch (DivideByZeroException)
    {
        Console.WriteLine("Man kan ikke dividere med 0");
    }
    finally
    {
           //Udføres også ved unhandled exceptions
        Console.WriteLine("Her kommer vi ALTID forbi");
    }
    Console.WriteLine("Her kommer vi NORMALT forbi");
}
```

Fejlhåndtering del 2: rapportering

- Exception-håndtering er delt i to:
  Nu Rapportering
  - try
  - catch (et antal)
  - finally



Fejlrapportering

- throw : Fejlrapportering sker et helt andet sted i programmet!

```
class person
{
    private string _name;

    public string Name
    {
        get { return _name; }
        set
        {
            if(value==null)
                throw new ArgumentNullException("value", "Name value cannot be null");
            _name = value;
        }
    }
}
```

Fejlrapportering

- throw : Fejlrapportering sker et helt andet sted i programmet!

```
class Person
{
    private string _name;

    public string Name
    {
        get { return _name;
        set
        {
            if(value==null)
                throw new ArgumentNullException("value", "Name value cannot be null");
            _name = value;
        }
    }
}
```

(<no parameters>)

(**string message**, Exception innerException)

(**string paramName**)

(**string paramName**, string message)

Initializes an instance of the **ArgumentNullException** class with a specified error message and the name of the parameter that causes this exception.
paramName: The name of the parameter that caused the exception.

Vigtige budskab omkring exceptions

- Exceptions skal **kun** bruges til fejlhåndtering, ikke control-flow
- Exceptions printer ikke noget ud – terminologi er vigtigt
- Throw afbryder control-flow: et jump til første passende catch
- Mere om exceptions senere i kurset
  - Findes i bogens kapitel 11

Exceptions

- **Exception:** En uventet/alvorlig fejl der sker under program udførelse
- Når exception *kastes* ophører normal kode-eksekvering øjeblikkeligt.
- Der genereres et exception objekt der indeholder information om fejlen.
- Hvis exception ikke håndteres eksplicit (*fanges*) returnerer metoden *øjeblikkeligt* og exception propageres tilbage igennem *call stakken* indtil den fanges.
- Hvis exception ikke håndteres i call stakken termineres programmet.
- Exceptions håndteres i try…catch…finally blokke.

```
try                       { /* exceptions kan ske */   }
catch (ExceptionA)    { /* håndter ExceptionA */  }
catch (ExceptionB b) { /* håndter ExceptionB */  }
finally { /* bliver altid udført - også ved uhåndterede exceptions*/ }
//mere kode – bliver kun udført hvis exceptions håndteres (eller udebliver)
```

Almindelige Exceptiontyper

Exception Type	Description
System.Exception	A generic exception from which other exceptions derive.
System.ArgumentException	A means of indicating that one of the parameters passed into the method is invalid.
System.ArgumentNullException	Indicates that a particular parameter is null and that this is not valid for that parameter.
System.FormatException	Indicates that the string format is not valid for conversion.
System.IndexOutOfRangeException	Indicates that an attempt was made to access an array element that does not exist.
System.InvalidCastException	Indicates that an attempt to convert from one data type to another was not a valid conversion.

Exceptionhierarki: : http://msdn.microsoft.com/en-us/library/aa904325(VS.71).aspx

Excerpted from Essential C# 4.0 by Mark Michaelis (ISBN: 0321694694).
Copyright 2010 Pearson Education, Inc.

Exceptions er informationsbærere

- Et exception objekt indeholder informative properties der beskriver en fejl i detalje:
  - **Message:** Kort beskrivelse af exception.
  - **Source:** Navn på program eller objekt der forårsagede exception.
  - **TargetSite**. Detaljer om metode der forårsagede exception.
  - **StackTrace**. Call stak der førte til exception. (Godt til testing/debugging)
  - **InnerException**. Når en exception kastes som resultat af en anden exception, kan den oprindelige exception gemmes I denne property. InnerException indeholder selv alle ovenstående properties.

- Specialiserede exception typer kan desuden inkludere yderligere relevant information.
- Fx.: ArgumentException inkluderer 'ParamName' property der giver detaljer om pågældende parameter.

Fang (catch) Exceptions

- C# tillader flere catch blokke.
  - Når exception kastes, udføres første catch blok der matcher exceptiontypen.
- Match bestemmes af nedarvningskæden fra Exception klassen.
- Catch blokke skal optræde fra mest specifikke (afledte) til mest generelle (hvis man ikke overholder reglen skal compileren nok brokke sig).

Catch-rækkefølge – fra specifik til generel

```
// Catching Different Exception Types
try
{
   // ... //
   throw new InvalidOperationException("Arbitrary exception");
}
catch (NullReferenceException exception)
{  /*  Handle NullReferenceException */   }
catch (ArgumentException exception)
{   /*  Handle ArgumentException */  }
catch (InvalidOperationException exception) //HER
{   /*  Handle ApplicationException */   }
catch (Exception exception)
{   /*  Handle Exception */   }
```

Kast en exception

- Runtime kan kaste exceptions, men det kan vi også selv med throw .
- Når ny exception kastes bliver nuværende metode starten på stack trace

```
public static int Parse(string textDigit)
{
   string[] digitTexts ={ "zero", "one", "two", "three", …, "eight", "nine");
   int result = Array.IndexOf(digitTexts, textDigit.ToLower());
   if (result < 0)
   {
      throw new ArgumentException(
         "The argument did not represent a digit", "textDigit");
   }
   return result;
}
```

Videresend exception

- Man kan fange en exception og derefter genkaste selvsamme exception (I modsætning til at kaste ny exception)
- Typisk til logging: Vi kan ikke håndtere fejlen, men vi noterer den.
- Forskel: Den genkastede exception bevarer sit stack trace.

```
try {...}
catch (Exception ex)
{
   //log…
   throw;  //vi genkaster samme exception
   //Brug IKKE: throw ex; //giver nyt stack trace
}
```

Guidelines for Exception Handling

- Fang kun exceptions der kan håndteres.
  - Overlad resten til metoder længere nede i kaldstakken
  - Skjul ikke exceptions der ikke FULDT håndteres.
- System.Exception og andre generelle catchblokke skal være sjældne.
  - Fristende at fange alt og fortsætte, men kritiske fejl kan blive begravet -> ustabilt system
- Undgå exception rapportering/logging for højt i kaldstakken
  - Hvis der logges og genkastes igennem callstakken får vi masser redundant info

Guidelines for **catch**
Exception Handling

- Brug throw istedet for throw <exception object> for at beholde stack trace.
  - Når vi kaster ny exception genstartes stack trace til indeværende metode (historie går tabt)
- Genkast kun ny exception i følgende situationer:
  - Ændring af exceptiontype synliggør problemet.
  - Private data er del af originale exception.
  - Oprindelig exceptiontype er for specifik til at call site kan håndtere den.

Exception eller if-håndtering

- Afhænger af:
  - **Hyppighed**: Hvor ofte forventer man hændelse?
    - Ofte: Brug *if* (exceptions tager længere at udføre)
    - Sjældent: Brug exception (mindre kode udføres i normal tilfældet)
  - **Alvorlighed**: Er vi i stand til at håndtere fejlen?

```
if (conn.State != ConnectionState.Closed) {
    conn.Close();
}
-- VS. --
try {
    ...
    conn.Close();
    ...
}
catch (InvalidOperationException)
{ rapportér eller håndter fejl}
```

Definition af Custom Exceptions

- Generelt er de indbyggede exception klasser at foretrække da de er velkendte/velforståede.
- En custom exception klasse kan bruges når speciel håndtering/handlinger er påkrævede.
- Custom exception skal arve fra System.Exception (direkte eller indirekte).
- Konvention: Brug “Exception” suffiks.
  - Og et dejligt langt navn!
- Tilbyd mindst 3 constructors:
  1. default constructor;
  2. constructor der tager en message string parameter;
  3. En constructor der tager en message string parameter og en inner exception parameter

Custom exceptioneksempel

```
class InvalidPrinterMarginsException : Exception
{
    public InvalidPrinterMarginsException() : base() {}
    public InvalidPrinterMarginsException(string s) : base(s) {}
    public InvalidPrinterMarginsException(string s, Exception ex)
        : base(s, ex) {}
}
class WordApp {
    void Print(){
        if (error)
          throw new InvalidPrinterMarginsException("Margins are too small");
        ...
    }
    static void Main(string[] args) {
        try {
            Print();
        }
        catch (InvalidPrinterMarginsException ex) {
            Console.WriteLine(ex.Message);
        }
    }
}
```

Opsummering

- ExceptionHandling:
  - Fange og kaste (nye eller gamle) exceptions
  - Exception guidelines
  - Egne eller indbyggede exceptions