COMPUTERWORLD

Undtagelseshåndtering i C#

I modsætning til C++ kan man i C# ikke skrive et program uden undtagelseshåndtering, så derfor har jeg skrevet denne guide til hvordan men bruger det. Egne undtagelser er også med.

Skrevet den **03. Feb 2009** af **visualdeveloper** I kategorien **Programmering / C#** |

Undtagelseshåndtering i C#

Grunden til at jeg har valgt at skrive en artikel om undtagelseshåndtering i C# er, at det er en meget vigtig ting. En undtagelse er når Windows genererer en fejl - når dit program udfører en "ulovlig" handling. Fejlene kan opstå på mange forskellige måder, men man kan dele dem op i to grupper - dem men selv er skyld i som fx almindelige programmeringsfejl, og dem man ikke selv kan kontrollere som fx hvis der ikke er nok hukommelse til at programmet kan køre videre.

try...catch...finally
Når man skal lave en undtagelseshåndtering i C# skal man bruge de tre nøgle ord try, catch og finally. Det stykke kode som skal kunne kaste en undtagelse, sætter man i en try-blok og koden der skal håndtere undtagelser fra try-blokken, anbringer man i en catch-blok, og til sidst finally som bruges til oprydning som SKAL klares inden programmet afsluttes.

Her er syntaksen for en undtagelseshåndtering i C#:

```
try
{
...kode der kan generere fejl
}
catch
{
...fejlhåndtering
}
finally
{
...oprydning
}
```

...dvs. at den kode der kan indeholde fejl, skrives i try-blokken, den kode der skal håndtere fejlene, skrives i catch-blokken og den kode der skal ryde op sættes i finally-blokken.

Her er et godt lille eksempel på brug af try...catch:

```
using System;
class Eksperten
{
  public static void Main()
  {
  int i;
```

```
  try
  {
    i++;
  }
  catch
  {
    Console.WriteLine("Fejl fundet !");
  }
  }
}
```

...da det i C# er forbudt at bruge ikke-initialiserede variabler. (dvs at *i* ikke indeholder nogen værdi) kan man ikke lægge 1 til *i* og der genereres en fejl. Men da vi har lavet en undtagelseshåndtering, kommer Windows ikke med nogen pop-up men den går til catch-blokken som udskriver "Fejl fundet !".

Mere specifikke fejl ?
Denne form for catch vi har brugt her er bare den simple...som fanger alle fejl.
Hvis vi ønsker at fange mere specifikke fejl som fx. en division med 0. Her er et eksempel:

```
using System;
class Eksperten
{
	public static void Main()
	{
		int x = 0;
		int div = 0;
		try
		{
			div = 100/x;
			Console.WriteLine("Den linje bliver kompileret, men ikke udført !");
		}
		catch(DivideByZeroException de)
		{
			Console.WriteLine("DivideByZeroException" );
		}
		catch
		{
			Console.WriteLine("Alle andre Exceptions" );
		}
		finally
		{
			Console.WriteLine("Finally-Blok");
		}
		Console.WriteLine("Resultatet er {0}",div);
	}
}
```

Her bruger vi DivideByZeroException som fanger alle fejl der opstår når der bliver divideret med 0. Alle andre fejl bliver håndteret i den anden catch-blok. Alle undtagelser arver fra System.Exception.

Brug af finally
Efter en try- og en catch-blok kommer finally blokken som bruges til oprydning. Et godt eksempel på det er når vi åbner en fil og læser fra den, så skal vi sikre os at filen bliver lukket igen. Her er et eksempel på det:

```
using System;
using System.IO;

public class Eksperten
{
  public static void Main()
  {
     StreamReader s = null;

     try
     {
     s = File.OpenText("c:\\fil.txt");
     while (s.Peek() > -1)
       Console.WriteLine(s.ReadLine());  //her læses der fra filen
     }
     catch
     {
       Console.WriteLine("Der opstod en fejl !");  //en fejlhåndtering
     }
     finally
     {
       if (s != null)
           s.Close();  //her lukkes filen hvis den stadig er åben
     }
  }
}
```

Objektorinterede undtagelser
Alt I C# er objektorinteret, og derfor er undtagelser også objekter med egenskaber. Nogle af disse egenskaber kan være godt at kende. Her er et eksempel på brug af dem:

```
using System;

public class Eksperten
{
  public static void Main()
  {
     string s = "Eksperten";

     try
     {
       int i = Int32.Parse(s);
     }

     catch (Exception e)
     {
       Console.WriteLine("Der er opstået en fejl");
```

```
      Console.WriteLine("Meddelelse: {0}", e.Message);
      Console.WriteLine("TargetSite: {0}", e.TargetSite);
      Console.WriteLine("Kilde: {0}", e.Source);
    }
  }
}
```

Her finder vi de tre egenskaber af objektet e (Exception). Message som viser fejlmeddelelsen, TargetSite som viser den metode hvor undtagelsen opstod og Source som viser applikationsnavnet.

I mit eksempel prøver vi at konvertere en String til en Integer men det kan vi ikke da en Integer kun kan indeholde heltal og derfor opstår der en fejl, som så bliver håndteret i catch-blokken.

Egne undtagelser - *NYT*
Ud over .NET-kernens undtagelser, kan du også lave eller kaste dine egne undtagelser som det jo hedder. Det gøres ved at bruge nøgleordet *throw* og nøgleordet *new*. Denne mulighed kan være praktisk hvis du ikke lige kan finde en exception-klasser der passer til dit program. Her er et eksempel på hvordan man kaster sine egne undtagelser:

```
using System;

public class Eksperten
{
  public static void Main()
  {
     try
     {
       string Tmp;
       DateTime Dato;

       Console.WriteLine("Indtast en dato, der ligger før dags dato");
       Tmp = Console.ReadLine();

       try
       {
          Dato = DateTime.Parse(Tmp);
       }
       catch
       {
          throw new Exception("Ikke en gyldig dato");
       }

       if (Dato >= DateTime.Now)
          throw new Exception("Dato skal ligge før dags dato");
          Console.WriteLine("Du har indtastet {0:d}", Dato);
       }
       catch (Exception e)
       {
         Console.WriteLine(e.Message);
       }
     }
}
```

Først kontrolleres om den indtastede dato kan konverteres til en gyldig dato. Hvis ikke det kan lade sig gøre, genereres en undtagelse med throw-nøgleordet, som her:

```
Indtast en dato, der ligger før dags dato
01-099-2006
Ikke en dato
```

Eller hvis datoen ligger før dags dato:

```
Indtast en dato, der ligger før dags dato
01-06-1999
Dato skal ligge før dags dato
Du har indtastet 01-06-1999
```

Det var så det...nu skulle du gerne kunne lidt mere...bare LIDT mere om exceptions i C#. Har du kritik, spørgsmål eller idéer til artiklen så skriv det i din anmeldelse eller i dette spørgsmål:
http://www.exp.dk/spm/656878
Skriver du en anmeldelse, herunder, så skriv lige en begrundelse !

Kommentar af skwat d. 18. Oct 2005 | 1

class hat{public static void Main(){Console.WriteLine("hat");}} er som jeg kan se et eksempel på et program uden error handling

Kommentar af brilleaben d. 18. Oct 2005 | 2

For kort - hvad med at beskrive hvordan egen exceptions kreeres og håndteres?

Kommentar af mysitesolution d. 19. Oct 2005 | 3

okay artikel... men hhm... har du ikke lært at fx i = i + 1; er langsommere end i++; og tilmed fylder det mere...

Kommentar af imago-dei d. 01. May 2006 | 4

God artikel. Jeg vil dog påpege at der er uenighed om hvordan exceptions skal behandles. Se. feks. denne artikel: http://codebetter.com/blogs/karlseguin/archive/2006/04/05/142355.aspx. Til kommentarerne nedenunder: jeg har lidt svært ved at tro at i = i +1 skal være langsommere runtime end i++. Begge dele bliver vel oversat til præcis det samme i compileren.

Kommentar af computerpeter d. 05. Nov 2005 | 5

Kommentar af ino d. 28. Mar 2006 | 6

God... Men kunne du ikke lave indryk i den kode du skriver for der er ret svært at læse...

Kommentar af iostream d. 20. Oct 2005 | 7

God artikel..jeg kan se at egne undtagelser kommer snart :-)
forresten er i = i + 1; rettet til i++; så nu er den god !

Kommentar af yezbarh d. 17. Oct 2005 | 8

Kort, men ellers fin

Kommentar af peter_bf d. 21. Oct 2005 | 9

Han har altså rettet det hele, og egne exceptions er også tilføjet så jeg kan ikke sige andet end top :D