# 14.0 Exceptions – undtagelser.

Når et Java-program afvikles, kan der opstå fejl, der får programmet til at luk­ke ned.

1) Det kan være fatale fejl som kommer udefra: Hukommelses-overflow eller fejl på den virtuelle Java-maskine (det program, hvor javapro­gram­mer­ne afvikles). Dem kan vi intet stille op overfor.

2) Der kan være fejl, der opstår under kørslen f.eks. pga. uhen­sigts­mæs­­sige bruger-indtastninger. Dem kan og bør vi håndtere, men vi er ik­ke tvunget til det.

3) Endelig kan der opstå fejl, fordi porte er optaget, filer ikke kan findes o.lign. Den type fejl **skal** vi håndtere. Allerede under kompileringen vil vi få fejl, hvis ikke vi tager hånd om dem.

I Java kaldes fejl *exception* og en *exception* er et objekt der kan ”kastes”, ”gri­bes” og sendes rundt mellem metoder og klasser. I denne lektion bruges det korte danske ord *fejl* men også ordet *undtagelse* bruges i stedet for *exception*. Der er ikke nogen dybere skelnen mellem *fejl* og *undtagelse*.

Java har et stort bibliotek over fejl, Exceptions and Errors, som er gengivet i større lærerbøger o.a.

Denne lektion er en kort introduktion, der viser principperne i fejlhåndtering og giver et par eksempler.

## 14.1 Nedluk og fejlmeddelelse

Her er et lille program, der forsøger at dividere med 0 (nul).

1 //Demonstrerer exceptions - undtagelser og den fejltekst java genererer.

2 //Programmet afbrydes pga fejl

3

4 public class DelMedNul

5 {

6 public static void main(String[] args)

7 {

8 int taeller = 10;

9 int naevner = 0;

10 System.out.println("\n\nNu er programmet startet\n\n");

11 System.out.println(taeller/naevner);

12 System.out.println("Denne linie ser vi aldrig");

13 }

14 }

Hvis du forsøger at køre programmet kompilerer det fint, men under kørslen sker dette:  
Programmet starter uden problemer, men så kommer en fejlmeddelelse.  
Hvis man kigger nærmere på den, kan man se at fejltypen er *ArithmeticException: / by zero* og at den er sket i linie 11 i *main*-metoden inde i klassen *DelMedNul*.   
Bemærk at linie 12 aldrig køres.  
Den trænede programmør kan straks tage fat på løsningen: Det vil vi også…

## 14.2 try- catch

Når en fejl opstår i Java, kastes der en exception – en undtagelse. Vi håndterer fejl ved at skrive den kritiske[[1]](#footnote-1) del af programmet inde i en try-blok. Derefter følger en eller flere catch-blokke som griber den specificerede fejl. Det er inde i catch-blokken at vi skal programmere, hvad der skal ske med fejlen.

public class DelMedNulogFang

{

public static void main(String[] args)

{

int taeller = 10;

int naevner = 0;

try

{

System.out.println("\n\nNu er programmet startet");

System.out.println(taeller/naevner);

System.out.println("Denne linie ser vi aldrig");

}

catch (ArithmeticException fejl)

{

System.out.println("\nEn fejl er grebet og vi "+

"bestemmer hvad der skal goeres:");

System.out.println("Fejlteksten er: " + fejl.getMessage());

System.out.println("Hvor er fejlen sket?"+

" (\"Call stack trace\") ");

fejl.printStackTrace();

}

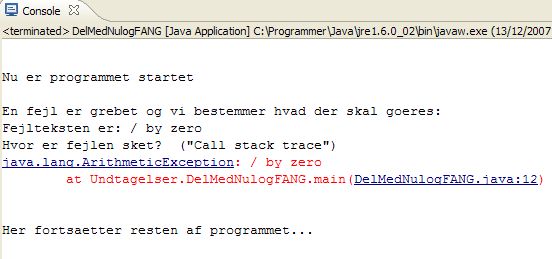
//Resten af programmet...

System.out.println("\n\nHer fortsaetter resten af programmet...\n\n");

}

}

På næste side ser du program-afviklingen.

  
Af fejlteksten fremgår det at fejlen opstår i linie 12.  
Bemærk at resten af *try*-blokken ikke udføres: Linie 13 udskrives aldrig.   
Men alt inde i *catch*-blokken udføres ligesom resten af programmet uden for *try-catch*.

## 14.3 try-catch med finally

Der findes en udvidelse af *try-catch*, der hedder *finally*. Den er tiltænkt kritiske sektioner, hvor det er nødvendigt at foretage sig meget vigtige ting før en eventuel nedlukning. Et eksempel kan være at afslutte telefon-forbindelse og lukke/gemme filer før nedlukning. På denne side er vist et primitivt eksempel med *finally*. Program-fejlen er dog ikke alvorlig nok til en rigtig nedlukning.

public class DelMedNulFinally

{

public static void main(String[] args)

{

try

{

int taeller = 10;

int naevner = 0;

System.out.println("\nNu er programmet startet\n");

System.out.println(taeller/naevner);

System.out.println("Denne linie ser vi aldrig");

}

catch (Exception fejl)

{ System.out.println("Hvad skal vi dog goere?"); }

finally

{

System.out.print("\nVigtige ting foretages i finally foer evt. nedluk");

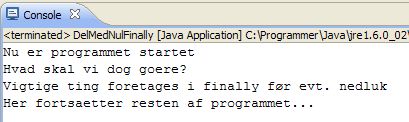
}

//Resten af programmet...

System.out.print("\n\nHer fortsaetter resten af programmet...\n\n");

}

}

Når programmet afvikles,  
ser det således ud:

## 14.4 Udbredelse / videre-formidling af fejl:

skaber

<<class>>

Udbredelse

<<class>>

UndtagelsesOmfang

niveau1()

{ try-catch }  
niveau2()

{ }  
niveau3()

{ **fejl sker** }

Hvis ikke en programmør får lagt en *try-catch* præcist der, hvor fej­len kan opstå, bliver fejlen sendt videre tilbage eller op i pro­gram­met, indtil der bliver gjort noget ved den.  
I det følgende eksempel skal vi se en undtagelse, som bliver sendt retur gennem flere metodekald. Skematisk kan det se således ud:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Oversat fra Propagation.java Author: Lewis and Loftus

// Demonstrates exception propagation.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public class Udbredelse

{ //-----------------------------------------------------------------

// Kalder niveau1-metoden for at påbegynde exception demonstration.

//-----------------------------------------------------------------

static public void main (String[] args)

{

UndtagelsesOmfang demo = new UndtagelsesOmfang();

System.out.println("\n\nProgrammet \"Udbredelse's\" main er startet");

demo.niveau1();

System.out.println("Programmet \"Udbredelse\" er loebet til ende.\n\n");

}

}

Klassen hvori fejlen opstår:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Oversat fraExceptionScope.java Author: Lewis and Loftus

// Demonstrates exception propagation.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public class UndtagelsesOmfang

{ //-----------------------------------------------------------------

// Griber og håndterer undtagelsen, der blev kastet på niveau3.

public void niveau1()

{

System.out.println("\nNiveau 1-metoden i \"UndtagelsesOmfang\" er startet.");

try

{

niveau2();

}

catch (ArithmeticException problem)

{

System.out.println ();

System.out.println ("Fejl meddelelsen er: " +

problem.getMessage());

System.out.println("Hvor er fejlen sket? (\"Call stack trace\"):");

problem.printStackTrace();

}

System.out.println("\nNiveau 1 i \"UndtagelsesOmfang\"slutter.");

}

//-----------------------------------------------------------------

// Nivau2 er et mellemniveau. Undtagelsen sendes igennem denne metode

// tilbage til niveau1-metoden.

public void niveau2()

{

System.out.println("Niveau 2 begynder...");

niveau3 ();

System.out.println("Niveau 2 slutter.");

}

//-----------------------------------------------------------------

// Udfører en beregning der producerer en fejl/undtagelse. Der bliver

// ikke gjort noget ved fejlen på dette niveau.

public void niveau3 ()

{

int taeller = 10, naevner = 0;

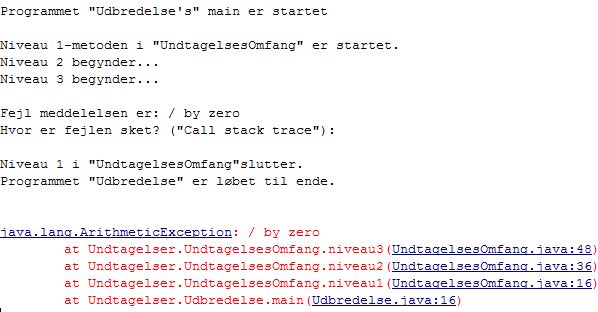
System.out.println("Niveau 3 begynder...");

int result = taeller / naevner;

System.out.println("Niveau 3 er slut.");

}

}

Når programmet kø­res, ser det således ud: Fejlen opstår inde i metoden niveau3( ).  
 Denne metode kø­res ikke til ende, ej hel­ler metoden niveau2, da den hel­ler ikke har nogen fejl­­­hånd­tering.   
 Først på niveau1 har vi en *try-catch* som tager hånd om problemet. Niveau1-metoden afsluttes korrekt og endelig *main*-metoden ovre i ”*Udbredelse*”.

Studér fejlmeddelelsen nøje. Følg fejlens vej op fra niveau3.

## 14.5 Throws

For fuldstændighedens skyld skal nævnes at en måde at omgås fejl på ( – især dem kompileren ikke vil oversætte uden fejlhåndtering) : I stedet for at bygge en *try-catch,* kan man nøjes med at erklærer at denne klasse kaster en fejl:

public class Udbredelse throws Exception

{

…

}

Det løser ingen problemer: Programmet vi stoppe, så snart en fejl opstår. Metoden må frarådes og kan kun forsvares, hvis et program er under udvikling og fejlhåndtering ikke er med i udviklings-etapen. Eller hvis programmet f.eks. senere skal kaldes af et andet program, som tager sig af fejlhåndteringen.

## 14.6 Læse og skrive til harddisk.

Hidtil har alle vores programmer kun eksisteret i computerens primære hukommelse (RAM). I dette afsnit vises eksempler på at læse en tekst og skrive en tekst på harddisken. Da vi skal kommunikere med hardware, er der alle muligheder for fejl. Disse fejl er af type 3 ( se indledningen til lektion 15): Fejl vi **skal** håndtere. Allerede under kompileringen vil vi få fejl, hvis ikke vi tager hånd om dem.  
 Som før nævnt findes et stort udvalg af fejl. I de følgende eksempler kan der ske fejl af typen IOException, som kompileren ikke vil tolerere uden fejlhåndtering.

1 import java.io.\*; // importer klasser til at kommunikere m.i/o  
 2  
 3 public class HentFil  
 4 {  
 5 public static void main(String[] args)  
 6 {  
 7 final int EOF= -1; //End Of File  
 8 int ind;  
 9 String filnavn;

10  
11 System.out.print("\nIndtast filnavn paa fil, der skal hentes: ");  
12 filnavn = Keyboard.readString();  
13 try  
14 {  
15 FileInputStream IndFil = new FileInputStream(filnavn);  
16 DataInputStream IndStroem = new DataInputStream(IndFil);  
17  
18 while ((ind = IndStroem.read()) != EOF)//Så længe fil ikke slut  
19 System.out.print((char)ind);  
20 IndFil.close();  
21 }  
22  
23 catch (IOException fejl)  
24 {  
25 System.out.println("\nFejlkode: " + fejl);  
26 }  
27 finally //Kan udelades  
28 {  
29 System.out.println("\nHvis noget vigtigt skal opryddes...\n");  
30 }  
31 System.out.println("\nHer kommer resten af programmet...\n");  
32 } // slut på main-metode  
33 } // slut på klassen Hentfil

I linie 15 og 16 omdannes data til en data-strøm. Populært sagt skal data læses i en lang seriel strøm af data. Det kræver et par kunstgreb at omdanne dem. Det vil føre for vidt at udrede dette i denne lektion, så indtil videre må I ”sluge kamelen”.

**Opgave 14.1**

Prøv at køre programmet. Afprøv både at indlæse en fil der eksistere og en der ikke eksistere. Hvad sker der? Du kan hente og gemme hvor du vil. Angiv blot stien *String filnavn = d:\Documents and Settings\ Ditnavn\ Dokumenter\ navnPaFil\_version2.txt*På næste side vises et eksempel på hvordan der kan skrive til en fil.

1 import java.io.\*;

2

3 public class SkrivFil  
 4{  
 5 public static void main(String[] args)  
 6 {

7 final int CR= 13; //Carrier return = Enter  
 8 int udKarakter;  
 9 String filnavn;  
10 System.out.print("\nIndtast filnavn paa fil,der skal skrives/overskrives i: ");  
11 filnavn = Keyboard.readString();  
12 try  
13 {  
14 FileOutputStream UdFil = new FileOutputStream(filnavn);  
15 DataOutputStream UdStroem = new DataOutputStream(UdFil);  
16 System.out.println("\nBegynd indtastning: ");  
17 while ((udKarakter = System.in.read()) != CR)//Så længe ikke "enter"  
18 UdStroem.write(udKarakter);  
19 UdFil.close();  
20 }  
21 catch (IOException fejl)  
22 {  
23 System.out.println("\nFejlkode: " + fejl);  
24 }  
25 System.out.println("\nHer kommer resten af programmet...\n");  
26 }

Til sidst vises et eksempel på en variation over ”Hentfil” som faktisk ikke håndterer fejlene. Men klassen proklamerer at den kan ”kaste” fejl af typen IOException. Det er så meningen at andre klasser skal tage sig af dem jævnfør afsnit 14.4 om udbredelse.

1 import java.io.\*;

2

3 public class HentFilThrows  
 4 {  
 5 public static void main(String[] args) throws IOException  
 6 {  
 7 final int EOF= -1; //End Of File  
 8 int ind;  
 9 String filnavn;

10  
11 System.out.print("\nIndtast filnavn paa fil, der skal hentes: ");  
12 filnavn = Keyboard.readString();  
13

14 FileInputStream IndFil = new FileInputStream(filnavn);  
15 DataInputStream IndStroem = new DataInputStream(IndFil);  
16

17 while ((ind = IndStroem.read()) != EOF)//Så længe fil ikke slut  
18 System.out.print((char)ind);

19 IndFil.close();

1. System.out.println("\nMon dette bliver udskrevet?");
2. }
3. }

**Opgave 14.2** Afprøv ”HentFilThrows”. Afprøv både at indlæse en fil der eksistere og en der ikke eksistere. Hvad sker der?   
 \_\_\_ \_\_\_\_\_ \_\_\_\_

1. Den kritiske del, er der, hvor en fejl kan opstå. F.eks. hvor der er brugerindtastninger. [↑](#footnote-ref-1)