Kursus 02199: Programmering
Exceptions og I/O: afsnit 8.1 – 8.4

Anne Haxthausen, IMM, DTU

1. Exceptions (afsnit 8.1)
2. Input og output (I/O) (afsnit 8.2-8.4)
3. 02199 opsummering

Fejl i programmer

Tre typer af fejl:

1. Syntaks fejl: opdages af `javac` compileren
2. Kørsels fejl (f.eks. division med 0): opdages af `java` fortolkeren og giver anledning til exceptions
3. Logisk fejl (programmet gør noget andet, end man havde tænkt sig): opdages måske ved test

En *exception* er hændelse, der sker, når der opstår fejl under udførelse af et program. Den bevirker, at programmet afbrydes på stedet, med mindre man laver *exception handling*.

Exceptions

Programmer skal kunne signalere og håndtere fejlsituationer — ved at kaste og håndtere *exceptions*.

- Hvad er en 'exception'?
- Hvordan kaster man en exception (**throw**-kommandoen)?
- Hvordan kan man håndtere en exception (**try-catch**-kommandoen) ?
- Hvornår skal man erklære en exception (med **throws**) i metode headere?
- Brugerdefinerede exception klasser.

Hvad er en exception (undtagelse)?

En undtagelse er et objekt af en subklasse af klassen `Throwable`.

Det bruges til at signalere og beskrive en fejlsituation, der opstår ved programkørsel.

Objektet indeholder:

- hvilken exception klasse, den er en instance af
- en meddelelse om arten af fejlen
- en beskrivelse af, hvor fejlen opstod

Denne information printes ved kald af objektets `printStackTrace` metode. Objektets `getMessage` metode returnerer meddelelsen som en String.

Her er en del af klassehierarkiet for `Throwable`, `Error`, og `Exception`:

```
Throwable
    Error                               (skal ikke erklaeres)
        VirtualMachineError
            OutOfMemoryError
            StackOverflowError
    Exception                           (skal erklaeres)
        IOException
            FileNotFoundException
        RuntimeException                (skal ikke erklaeres)
            ArithmeticException
            IndexOutOfBoundsException
            IllegalArgumentException
                NumberFormatException
```


Kastning af exceptions

Når en runttime fejl opstår under udførelse af et program, *kastes* en exception.

To slags runttime fejl:

- predefinerede: exception kastes automatisk
- brugerdefinerede: exception skal kastes eksplicit vha. orden **throw**

Eksempel på automatisk (predefineret) kastning af exception

Ved kørsel af

```
public class ArrayIndex  {
  public static void main (String[] args) {
         int[] a = {1,2,3,4,5};
         System.out.println(a[5]); //fejlindeksering
         System.out.println("Will not print");
  }
}
```

afbrydes programmet med en fejlmeddelelse:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
    at ArrayIndex.main(ArrayIndex.java:4)
```


Explicit kastning af en undtagelse med ordren throw

Nedenstående Dato-konstruktor tillader konstruktion af ulovlige datoer, f.eks. konstrueres 30/13-1999 med `Dato(1999, 13, 30)`

```
Dato(int aar, int maaned, int dag) {
  this.aar = aar; this.maaned = maaned; this.dag = dag; }
```

En bedre løsning: konstruktoren kaster en undtagelse, hvis den bliver bedt om at lave en ulovlig dato:

```
class Dato {
  private int aar, maaned, dag;

  Dato(int aar, int maaned, int dag) throws Exception {
    if (ok(aar, maaned, dag))
      { this.aar = aar; this.maaned = maaned; this.dag = dag; }
    else
      throw new Exception("Ulovlig dato: "
                          + aar + " " + maaned + " " + dag);
  }
...
}
```

Ved kørsel af

```
public class Datoexn01 {
    public static void main(String[] args) throws Exception {
      Dato d1 = new Dato(1999, 13, 30); //ulovlig dato
      System.out.println(d1);
    }
}
```

afbrydes programmet med en fejlmeddelelse:

```
Exception in thread "main" java.lang.Exception: Ulovlig dato: 1999 13 30
    at Dato.<init>(Dato.java:8)
    at Datoexn01.main(Datoexn01.java:3)
```


Håndtering af exceptions

Måder at behandle en exception på:

1. ignorer den som i forrige eksempler
2. fang den (med **try-catch**)
   (a) hvor den opstår
   (b) et andet sted i programmet.

Løsning 1 kan give udemærkede fejlmeddelelser, men programmet går ned midt i sin udførelse, hvilket kan være uheldigt.

Løsning 2 giver mulighed for, at programmet ikke går ned ved fejl, men bliver bragt på benene igen ved at foretage sig noget specielt.

Hvis man bruger løsning 2b kan man opsamle fejl fra flere dele af ens program på et sted — godt, hvis man er interesseret i at fange fejl for en del af programmet, men er ligeglad med præcist hvor fejlen opstod.

Ad 1: uhåndterede exceptions

Når en exception ignoreres i programmet sker følgende:

1. programmet stopper.
2. der udskrives en besked indeholdende information om:
   (a) hvilken exception der opstod
   (b) hvor i programmet den opstod
   - første linje indeholder navnet på den metode, hvor exceptionen opstod samt evt meddelelse
   - de resterende linjer viser hvilke metoder, man har kaldt for at nå til metoden

Ad 2: at fange exceptions med try-catch ordren

```
try
{
  ordrer1
}
catch (Exception exn)
{
  ordrer2
}
```

Der sker følgende:

- Ordrerne i *ordrer1* udføres.
- Kastes der en exception `e1` under udførelsen af *ordrer1*, så udføres *ordrer2*.
- Variablen `exn` er bundet til det kastede exception-objekt under udførelsen af *ordrer2*.
- Kastes der ikke en exception under udførslen af *ordrer1* ignoreres *ordrer2*.

Ad 2: eksempel på at fange en undtagelse

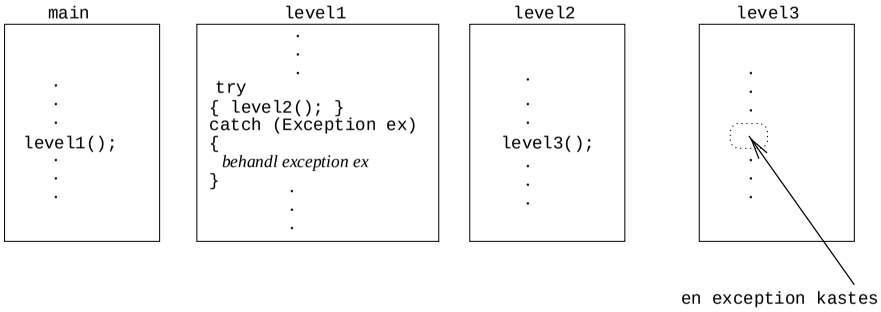
Kørsel af

```
public class Datoexn1 {
  public static void main(String[] args) {
    try {
      Dato d1 = new Dato(1999, 13, 30);
      System.out.println(d1);
    }
    catch (Exception e) {
      System.out.println("Ulovlig dato!");
    }
    System.out.println("Hertil kommer vi!");
  }
}
```

giver uddata:

```
Ulovlig dato!
Hertil kommer vi!
```


Ad 2b: exception propagation



Erklæring af exceptions

Undtagelserne `Error` og `RuntimeException` og deres subklasser er 'unchecked'. Alle andre undtagelser er 'checked'.

For 'checked' exceptions gælder:

Hvis, der for en metode `m` er risiko for, at når den kaldes med `m(...)`, så kastes en checked exception af klasse `E` uden at den fanges inde i kroppen, så skal den erklæres i hovedet af metodeerklæringen med **throws** `E`.

Eksempel: se `Datoexn01`

For 'unchecked' exceptions gælder, at man aldrig erklærer dem.

Brugerdefinerede exception klasser

Man kan også lave sine egne exception klasser.

For et eksempel: se bogens listing 8.6.

I/O i Java

- Streams
- Input fra tastaturet og output til skærm
- Input fra og output til tekst filer

Streams

I Java bruger man *input* og *output* streams (da. strømme) til at kommunikere med omgivelserne: filer, keyboardet, etcetera.

Java har en familie af klasser til at lave stream objekter. Se bogens figur 8.3.

Streams kan opdeles i:

- input streams, der har metoder til at læse data fra omgivelserne.
- output streams, der har metoder til at udskrive data til omgivelserne.

og i

- tegn (tekst) streams, der fortolker de enkelte bytes som tegn
- byte (binære) streams

Går der noget galt i forbindelse med I/O, kastes der en `IOException`.

Indlæsning fra keyboard og udskrift på skærm

Vi har tidligere lært:

- *indlæsning fra keyboard (skærm)* vha. `Keyboard` klassens metoder
- *udskrift på skærm* vha. metoderne `System.out.print` og `System.out.println`

Eksempel:

```
public static void main(String[] args)  {
  int x;
  System.out.print("Indtast␣et␣tal:␣");
  x = Keyboard.readInt();
  System.out.println("Tallet␣er:␣" + x);
}
```

```
Indtast et tal: 13
Tallet er: 13
```


Tekst-filer versus binære filer

Alle filer består af bytes. 1 byte = 8 bits. En bit er `0` eller `1`.

Men det er forskelligt, hvordan de skal fortolkes.

Eksempel:

En fil med fire bytes:

01001010	01100001	01110110	01100001

Hvis det er en **tekstfil**, hvor hver enkel byte fortolkes som et ascii tegn, så repræsenterer de fire bytes følgende tegn:

J	a	v	a

Men hvis det er en **binær** fil, kan de fortolkes som noget andet, f.eks. et enkelt heltal (af typen `int`).

Tekst-filer

Definition: en tekst-fil er en tegnfølge, lagret på disk eller diskette.

Typiske filnavne: `score.txt, numbers.txt, Time.java,` ...

Man kan gemme både ord og tal i tekstfiler.

Eksempel: Tekstfilen `score1.txt`

```
Joe 3 4
John 4 5 6
```

består af 20 tegn:

J	o	e		3		4	\n		J	o	h	n		4		5		6	\n

Ide ved læsning af tekstfiler

Ved læsning af tekstfiler er det ofte mere interessant at læse hele ord og tal fremfor de enkelte bogstaver.

Ofte er vi ligeglade med *white space*: blanktegn, linjeskift og tabulatortegn.

De enkelte tal og ord kaldes *tokens* (da. brikker).

White space adskiller ofte de enkelte tokens.

Den første linie i `score1.txt` har 3 tokens:

Joe	3	4

Den anden linie i `score1.txt` har 4 tokens:

John	4	5	6

Læsning af tekstfiler, trin for trin

1. Åbning af filen `score1.java` og omdannelse til en buffered reader:

   ```
   FileReader r = new FileReader("score1.txt");
   BufferedReader in = new BufferedReader(r);
   ```

2. Man kan nu læse en linie ad gangen gangen fra "score1.txt"med `in.readline()`, der returnerer en `String`:

   ```
   String line = in.readline();
   ```

3. Læsning af en enkelt linie token for token:

   ```
   StringTokenizer tokenizer = new StringTokenizer(line);
   String token = tokenizer.nextToken();
   ```

4. Konverter `token` til ønsket type: f.eks. til `int` med `Integer.parseInt(token);`

Det hele pakkes kan pakkes ind i en **try-catch** til at fange exceptions.

Eksempel på læsning af tekstfil

Opgave:

Hver linie i filen `score1.txt` indholder først navnet på en golfspiller og derefter antal slag brugt per hul.

Disse data skal indlæses og på skærmen skal for hver spiller udskrives navn, antal huller og antal slag i alt:

```
Joe has played 2 holes and used 7 strokes
John has played 3 holes and used 15 strokes
```

Løsning: se `Laesning.java`, som kan downloades fra www-siden med foils.

linie for linie læser vi først navnet, dernæst scoren for de huller spilleren har spillet.

Skrivning til tekstfil

```
public static void main(String[] args) throws IOException {

  // opsaet output stream med navn "nn" (f.eks. "res.txt")
  FileWriter wri = new FileWriter("nn");
  PrintWriter out = new PrintWriter(wri);

  //udskriv til filen "nn"  med metoderne out.print og out.println
  ...

  out.close(); //luk filen efter brug
}
```

`out` ovenfor virker på samme måde som vores gamle ven `System.out` — d.v.s. vi kan f.eks. skrive `out.println("Eric");` eller `out.print(42);`.

02199 Opsummering

1. del af kurset (E01):

- Java programmering
- datalogiske grundbegreber:
  - typer og værdier
  - variable
  - sætninger og udtryk
  - begreber fra OOP: klasser, objekter, metoder, nedarvning
  - I/O of filer
  - exception handling
  - programmeludviklingsfaser
  - mm.

2. del af kurset (7. - 25. januar):

- grafiske brugergrænseflader (GUI)
- projekter

Plan for resten af dette semester

Øvelser:

- 13/11: opgaver i nedarvning
- 20/11: opgaver i I/O og exception handling
- 27/11: gamle eksamensopgaver
- 4/12: gamle eksamensopgaver

Brug den ekstra tid, når der ikke er forelæsninger, til at samle op på gamle opgaver.

Råd vedr. eksamen

Forbered jer:

- Læs stoffet (inkl. overheads).
- Regn de gamle opgaver (stil evt. dig selv små opgaver).
- Gør dig klart, hvad det er, du har lært.
- Lav nogle “opskrifter” til dig selv.
- Hvis du har spørgsmål til opgaver eller bog, så stil det til din hjælpelærer.
- Hvis du har spørgsmål af generel karakter, så stil det til mig.

Ved eksamenen:

- Kig først hele eksamenssættet igennem.
- Start med den opgave, du synes bedst om.
- Læs opgaveteksten grundigt og svar på det, der bliver spurgt om.
- Hvis du kører kast, så gå videre til næste spørgsmål.

HELD OG LYKKE!