

Håndtering af forudsigtelige fejltilstande

- En undtagelse ('exception') er en special værdi (et objekt)
 - Rejsning ('throwing') af en undtagelse
 - Håndtering ('catch') af en undtagelse
- Lewis og Loftus, afsnit 8.1.

Systematisk afprøvning af programmer

- Indkøring ('debugging') versus afprøvning ('test') af programmer
- Intern afprøvning
- Ekstern afprøvning

Noter: *Systematic Software Test*

Howdan kan man signalere at noget går galt under en beregning?
Fire muligheder:

- (a) Afbryd programmet og skriv en fejlmeddelelse på skærmen.
- (b) Sæt en global logisk variabel `Fejl til true` eller lignende.
- (c) Returnér en fejlværdi (fra udtrykket eller metoden).
- (d) Rejs en undtagelse.

Vurdering:

- Med mulighed (a) kan man ikke lave robuste og brugervenlige systemer.
 - Ved mulighed (b) skal man altid huske at tjekke fejlvariablen.
 - Mulighed (c) duer for metoder, men ikke for konstruktorer, der jo altid skal lave et veiformet objekt.
 - Mulighed (d) duer kun hvis der er en 'ledig værdi' såsom `-1` eller `null` som kan bruges til at signalere fejl.
- Javas idé om undtagelser stammer fra C++, der har ideen fra Standard ML.

Ude fra kommende fejlsituationer som man skal tage hensyn til

Der er altid noget der kan gå galt. Man kan komme til:

- at dividere med nul (program `Error1.java`);
- at bruge et ulovligt indeks i en tabel (program `Error2.java`);
- at åbne en ikke-eksisterende fil (program `Error3.java`);
- at skrive en metode der kalder sig selv i det uendelige (program `Error4.java`);
- at bruge al maskinens lager (program `Error5.java`);
- at læse et heltal fra en ikke-veiformet tegnstring (program `Error6.java`);
- ...

Det er vigtigt:

- (1) at programmet bliver gjort opmærksom på problemet;
- (2) at programmet får mulighed for at håndtere (afhjælpe) problemet.

Hvad er en undtagelse (exception)?

En undtagelse er et objekt af en klasse der implementerer `Throwable` (som er et interface).

En undtagelse er en meddelelse om at noget gik galt.

Dél af klasserækkel for `Throwable`, `Error`, og `Exception`:

<code>Throwable</code>	
<code>Error</code>	(uærklærede)
<code>VirtualMachineError</code>	
<code>OutOfMemoryError</code>	
<code>StackOverflowError</code>	
<code>Exception</code>	(skal erklæres)
<code>IOException</code>	
<code>FileNotFoundException</code>	
<code>FileNotFoundException</code>	(uærklærede)
<code>RuntimeException</code>	
<code>ArithmeticException</code>	
<code>IndexOutOfBoundsException</code>	
<code>IllegalArgumentException</code>	
<code>NumberFormatException</code>	

At rejse en undtagelse: `orden throw`

Motivation: En `Dato`-konstruktor bør kun lave `Dato`-objekter der svarer til lovlige datoer.

Hvad hvis `Dato`-konstruktoren bliver bedt om at lave den ulovlige dato 30/13-1999?

Konstruktoren skal returnere et `Dato`-objekt. Den kan f.eks. ikke returnere `null`.

En (dårlig) mulighed er at lave og returnere et objekt med 'lovliggjorte' felter:

```
Dato(int aar, int maaned, int dag) {
    if (ok(aar, maaned, dag))
        { this.aar = aar; this.maaned = maaned; this.dag = dag; }
    else
        { this.aar = aar; this.maaned = 12; this.dag = 24; }
}
```

Men det er forvirrende: Hvordan skelne mellem rigtige juleaften-datoer, og så dem der betegner fejl?

Det er bedre at rejse en undtagelse (fil `Datoexn1.java`):

```
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();
}
```

Eksempel (`Datoexn1.java`)

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

En undtagelses `toString()`-metode udskriver undtagelsen, hvis den ikke håndteres.

Hvis man giver `Exception`-konstruktoren et argument, så udskrives det også.

`Dato`-konstruktoren kunne give mere information om fejlen på denne måde:

```
throw new Exception(aar + " " + maaned + " " + dag);
```

At håndtere (eller fange) en undtagelse: `orden try-catch`

```
try { ordre1 }
catch (Exception e) { ordre2 }
```

Ordrene i `ordrer1` udføres.

Hvis der rejses en undtagelse `e1` under udførelsen af `ordrer1`, og `e1` tilhører `Exception` eller en subklasse af `Exception`, så udføres `ordrer2`.

Variablen `e` er da bundet til undtagelses-objektet `e1` under udførelsen af `ordrer2`.

Hvis ikke der rejses en undtagelse under udførelsen af `ordrer1` så ignoreres `ordrer2`.

```
try {
    ordre_1
    ...
    ordre_n
} catch (Exception e) {
    ordrer_2
}
// e sættes lig e1 før ordrer_2 udføres
```

Et andet eksempel (`ValutaGUI9.java`)

```
class BeregnLytter implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        try {
            int eu = Integer.parseInt(argument.getText());
            resultat.setText(Double.toString(eu * 7.42));
        } catch (NumberFormatException exn) {
            resultat.setText("Fejl: skriv et heltal!");
        }
    }
}
```

Hvis `argument.getText()` ikke udgør et velformet heltal, så rejses `NumberFormatException`.

Programmet håndterer undtagelsen med `try-catch`.

Det skriver en informativ meddelelse i resultat-vinduet i stedet for en ret uforståelig meddelelse i DOS-vinduet.

Erklærede og uerklærede undtagelser (Error 3 - Java)

Undtagelserne `Error` og `RuntimeException` og deres subclasses er *uerklærede* ('unchecked').

Alle andre undtagelser skal erklæres (er 'checked'):

Hvis en sådan undtagelse kan undslippe en metode, skal den erklæres i metodehovedet:

```
public static void main(String[] args) throws FileNotFoundException {
    InputStream is = new FileInputStream("glyf.paf");
}
```

Dette gør det klart for omverdenen at metoden kan fejle.

`Error` og `RuntimeException` kan rejses næsten hvor som helst.

Det ville være meget besværligt at skulle erklære dem alle vegne. Derfor er de uerklærede ('unchecked').

Indkøring versus afprøvning af programmer

Indkøring ('debugging')

Usystematisk.

Formål: få programmet til at køre.

Inddatasæt: små eksempler, gerne realistiske.

Afprøvning ('software test')

Systematisk.

Formål: at afsløre fejl i programmet (så de kan blive rettet).

Inddatasæt: omhyggeligt konstrueret specielt til dette formål.

Systematisk afprøvning er kvalitetsikning.

Nu til noget helt andet: Systematisk afprøvning

Ikke-tvivelde programmer er næsten altid defekte.

Fejl i programmer kan have alvorlige konsekvenser:

- I Golf-krigen (1991) ramte nogle amerikanske Patriot-missiler ved siden af indkommende irakiske Scud missiler (som derfor dræbte nogle snese mennesker på jorden).
 - Fejl i styreprogrammer i bagagehåndteringssystemet i Denver International Airport forsinkede lufthavnens åbning med ét år (1995), hvilket gav et tab på ca. 360 millioner US dollar.
 - Den første opsendelse af den europæiske Ariane 5 raket fejlede (1996), hvilket gav et tab på ca. 1 milliard US dollar. (Programfejl og mangelfuld afprøvning).
 - Fejl i dårligt designede styreprogrammer til Therac-25 strålebehandlingsudstyr (1987) udsatte adskillige kræftpatienter for store strålingsdoser, og dræbte nogle af dem.
- Vores programmer er simple og forårsager mindre ulykker. Men programmerne bør være fejlfri alligevel.

Man kan sikre fejlfri programmer ved at bevise at programmerne er korrekte (med løkkeinvarianter osv).

Det bruges f.eks. til Metro-styreprogrammer (Frankrig), diverse rumudstyr, ...

Men det er ofte for dyrt, for mandsskabskrævende, eller for tidskrævende.

Desuden sikrer det kun mod visse typer fejl. F.eks. ikke mod dårlige brugergrænseflader.

Intern afprøvning og ekstern afprøvning

	Intern afprøvning	Ekstern afprøvning
Udgangspunkt	programteksten	problemmstillingen
Fundne fejltyper	logikfejl	oversete tilfælde
	fejlinitialiserede variable	oversete krav

Intern afprøvning kaldes også strukturel afprøvning.

Ekstern afprøvning kaldes også funktional afprøvning.

I begge tilfælde skal afprøveren opstille en *afprøvning* ('test suite') med:

- en tabel over inddataegenskaber
- en tabel over inddatasæt og de tilsvarende forventede uddata

For at afprøve programmet køres det med inddatasættene, hvorefter man sammenligner de faktiske uddata med de forventede uddata.

Intern afprøvning: Valg- og gentagelsesordrer

Mål 1: alle dele af programmet skal have været udført.

Mål 2: alle grene af valg- og gentagelsesordrer har været udført.

Ordtype	Tilfælde der skal afprøves
if	Betingelse falsk og sand
for	Nul, ét, og flere gennemløb
while	Nul, ét, og flere gennemløb
do-while	Ét og flere gennemløb
switch	Hver gren skal have været udført

Intern afprøvning: Sammensatte logiske udtryk

Afprøv alle mulige kombinationer af leddenes sandhedsværdier.

Konjunktion (og, &&):

$(x \neq 0) \ \&\& \ (1000/x > y)$

giver

$(x \neq 0)$	$\&\&$	$(1000/x > y)$
falsk		falsk
sand		falsk
sand		sand

Disjunktion (eller, ||):

$(x == 0) \ || \ (1000/x > y)$

giver

$(x == 0)$	$ $	$(1000/x > y)$
sand		falsk
falsk		falsk
falsk		sand

Intern afprøvning, eksempel 1: find mindste og største tal (Minmax.java)

```
public class Minmax {
    public static void main ( String[] args ) {
        int mi, ma;
        if (args.length == 0)
            System.out.println("No numbers");
        else {
            mi = ma = Integer.parseInt(args[0]);
            for (int i = 1; i < args.length; i++) {
                int obs = Integer.parseInt(args[i]);
                if (obs > ma) ma = obs;           /* 3 */
                else if (mi < obs) mi = obs;     /* 4 */
            }
            System.out.println("Minimum = " + mi + "; maximum = " + ma);
        }
    }
}
```

Tablet over afprøvningsstifælde

Valg	Inddatasæt	Inddataegenskab
1 sand	A	Ingen tal
1 falsk	B	Mindst ét tal
2 nul gange	B	Netop ét tal
2 én gang	C	Netop to tal
2 flere gange	E	Mindst tre tal
3 sand	C	Tal > hidtidigt maksimum
3 falsk	D	Tal ≤ hidtidigt maksimum
4 sand	E tal 3	Tal ≤ hidtidigt maksimum og > hidtidigt minimum
4 falsk	E tal 2	Tal ≤ hidtidigt maksimum og ≤ hidtidigt minimum

Tabel over inddatasæt og forventede uddata

Inddatasæt	Iddata	Forventede uddata
A	(tom)	'No numbers'
B	17	17 17
C	27 29	27 29
D	39 37	37 39
E	49 47 48	47 49

Føllj

Inddatasæt D og E giver forkørte uddata, henholdsvis 39 39 og 49 49.

Forkert betingelse ved 4.

```

Retet program
public class Minmax {
    public static void main ( String[] args ) {
        int mi, ma;
        if (args.length == 0)
            System.out.println("No numbers");
        else {
            mi = ma = Integer.parseInt(args[0]);
            for (int i = 1; i < args.length; i++) {
                int obs = Integer.parseInt(args[i]);
                if (obs > ma) ma = obs;
                else if (obs < mi) mi = obs;
            }
            System.out.println("Minimum = " + mi + " ; maximum = " + ma);
        }
    }
}

```

Rettede afprøvningstilfælde

Valg	Inddatasæt	Inddataegenskab
1 sand	A	Ingen tal
1 falsk	B	Mindst ét tal
2 nul gange	B	Netop ét tal
2 én gang	C	Netop to tal
2 flere gange	E	Mindst tre tal
3 sand	C	Tal > hidtidigt maksimum
3 falsk	D	Tal ≤ hidtidigt maksimum
4a sand	E tal 2	Tal ≤ hidtidigt maksimum og < hidtidigt minimum
4a falsk	E tal 3	Tal ≤ hidtidigt maksimum og ≥ hidtidigt minimum

De gamle inddatasæt kan genbruges.

```

Intern afprøvning, eksempel 2: find to mindste tal (måske ens) (MinTwo.java)
public static void main (String[] args) {
    int mi1 = 0, mi2 = 0;
    if (args.length == 0)
        System.out.println("No numbers");
    else {
        mi1 = Integer.parseInt(args[0]);
        if (args.length == 1)
            System.out.println("Smallest = " + mi1);
        else {
            int obs = Integer.parseInt(args[1]);
            if (obs < mi1)
                { mi2 = mi1; mi1 = obs; }
            for (int i = 2; i < args.length; i++) {
                obs = Integer.parseInt(args[i]);
                if (obs < mi1)
                    { mi2 = mi1; mi1 = obs; }
                else if (obs < mi2)
                    mi2 = obs;
            }
            System.out.println("The two smallest are " + mi1 + " and " + mi2);
        }
    }
}

```

Tabel over afprøvningsstiftæalde

Valg	Inddatasæt	Inddatabegenskab
1 sand	A	Ingen tal
1 falsk	B	Mindst ét tal
2 sand	B	Netop ét tal
2 falsk	C	Mindst to tal
3 falsk	C	Andet tal \geq første tal
3 sand	D	Andet tal < første tal
4 nul gange	D	Netop to tal
4 én gang	E	Netop tre tal
4 flere gange	H	Mindst fire tal
5 sand	E	Tredje tal < hidtil mindste
5 falsk	F	Tredje tal \geq hidtil mindste
6 sand	F	Tredje tal \geq hidtil mindste og < næstmindste
6 falsk	G	Tredje tal \geq hidtil mindste og \geq næstmindste

Tabel over Inddatasæt

Inddatasæt	Inddata	Forventede uddata
A	(tom)	'No numbers'
B	17	17
C	27 29	27 29
D	39 37	37 39
E	49 48 47	47 48
F	59 57 58	57 58
G	67 68 69	67 68
H	77 78 79 76	76 77

Fejl!

Inddatasæt C giver forkerte resultater: 27 og 0.

Variablen mi 2 får ikke tildelt nogen værdi før den skrives ud (i tilfælde C).

Den beholder altså sin begyndelsesværdi, 0.

En passende tilfælding til mi 2 er nødvendig.

Rettet program

```
public static void main (String[] args) {
    int mi1 = 0, mi2 = 0;
    if (args.length == 0)
        System.out.println("No numbers");
    else {
        mi1 = Integer.parseInt(args[0]);
        if (args.length == 1)
            System.out.println("Smallest = " + mi1);
        else {
            int obs = Integer.parseInt(args[1]);
            mi2 = obs;
            if (obs < mi1)
                if (mi2 = mi1; mi1 = obs; )
                    for (int i = 2; i < args.length; i++) {
                        obs = Integer.parseInt(args[i]);
                        if (obs < mi1)
                            { mi2 = mi1; mi1 = obs; }
                        else if (obs < mi2)
                            mi2 = obs;
                    }
            System.out.println("The two smallest are " + mi1 + " and " + mi2);
        }
    }
}
```

Afprøvning i praksis

Genbrug ikke dataværdier (eller forventede resultater) i de forskellige inddatasæt.

Gen inddatasættene så kan de nemt køres igen, f.eks. i en fil testminmax.dat:

```
java Minmax 17
java Minmax 27 29
java Minmax 39 37
java Minmax 49 47 48
```

Opsaml uddata i en tekstfil testminmax.res. Så kan resultatet af testkørsler sammenlignes automatisk (med diff under Unix/Linux eller fc under MS DOS).

Eller skriv særlige testklasser, som kalder de metoder der skal afprøves, og tjekker resultaterne.

Genkør afprøvningen efter hver ændring eller forbedring i programmet.

Afprøvning af visuelle brugergrænseflader (med vinduer og mus) er besværlig:

Den skal foretages manuelt, den kan ikke genkøres automatisk

Sammenfatning om intern afprøvning

Intern afprøvning kommer systematisk ud i krogene.

Tidskrævende, men nyttig aktivitet for udviklede programmer.

Ekstern afprøvning: Løser programmet opgaven?

Mål: undersøg om programmet løser den stillede opgave.

Metode: forsøg at påvise at programmet *ikke* løser opgaven.

Forudsætninger for ekstern afprøvning

1. Opgaven er nogenlunde præcist formuleret.
2. Afprøveren har en fornemmelse for 'vanskelige' tilfælde og forkerte måder at løse problemet på.
3. Man kan beregne eller sjusse sig frem til de forventede uddata uden programmets hjælp.

Opstilling af en ekstern afprøvning kan afsløre uklarheder i problemformuleringen.

Opstilling af en ekstern afprøvning kan være en god start på programmeringsarbejdet.

Tabel over inddataegenskaber

Inddatasæt	Inddataegenskab
A	Ingen tal
B	Netop ét tal
C1	To tal, ens
C2	To tal, stigende orden
C3	To tal, faldende orden
D1	Tre tal, stigende orden
D2	Tre tal, faldende orden
D3	Tre tal, største i midten
D4	Tre tal, mindste i midten

Tabel over inddatasæt og forventede uddata

Inddatasæt	Inddata	Forventede uddata
A	(tom)	Fejlmelding
B	17	17 17
C1	27 27	27 27
C2	35 36	35 36
C3	46 45	45 46
D1	53 55 57	53 57
D2	67 65 63	63 67
D3	73 77 75	73 77
D4	89 83 85	83 89

Ekstern afprøvning, eksempel 1: find mindste og største tal

Givet en (muligvis tom) række tal, find det største og det mindste af disse tal.

Uklarhed: Hvad skal man gøre med en tom liste af tal?

Afklaring: Vi antager der skal gives en fejlmelding.

(Hvilken anden logisk mulighed er der?)

Ekstern afprøvning, eksempel 2: Find største forskel

Givet en (muligvis tom) række tal, find den største forskel mellem to på hinanden følgende tal.

Uklarhed: Hvad med en række der indeholder nul eller ét tal?

Afklaring: Vi antager at der skal gives en fejlmeldelse.

Uklarhed: Største forskel med eller uden fortegn?

Afklaring: Vi antager det skal være den numeriske forskel (uden fortegn).

Inddatasæt	Inddataægenskab
A	Ingen tal
B	Netop ét tal
C1	To tal, ens
C2	To tal, stigende orden
C3	To tal, faldende orden
D1	Tre tal, stigende forskel
D2	Tre tal, faldende forskel

Tabel over Inddataægenskaber

Inddatasæt	Inddata	Forventede uddata
A	(tom)	Fejlmelding
B	17	Fejlmelding
C1	27 27	0
C2	36 37	1
C3	48 46	2
D1	57 56 59	3
D2	69 65 67	4

Tabel over Inddatasæt og forventede uddata

Intern versus ekstern afprøvning

Intern og ekstern afprøvning giver ofte anledning til de samme inddatasæt.

Fordele ved intern afprøvning

'Mekanisk', kræver systematik men ikke meget indsigt i problemet.

Finder logiske fejl i programmet.

Giver afprøveren (eller programmøren) anledning til at studere programmet indgående.

Kan føre til at omprogrammering og dermed et bedre program.

Afdækker alle problemløsningsdetaljer.

Fordele ved ekstern afprøvning

Uafhængig af programmet.

Skal ikke ændres når programmet ændres (men når opgaven ændres).

Giver afprøveren anledning til at studere problemformuleringen indgående.

Kan føre til omformulering og præcisering af problemformuleringen.

Ekstern afprøvning, eksempel 3: Lovlige datoer

Givet en dato *dtag* og et månedsnummer *md*, afgør om de bestemmer en lovlig dato i et ikke-skudår.

Dagen og måneden angives som heltal.

Eksempler: 31/12 og 31/8 er lovlige datoer, men 29/2 og 1/13 er ulovlige

	Inddatasæt	Inddata	Forventede uddata
A	0 1	01.01.19	
	1 0	01.01.19	
	1 1	01.01.19	
	1 1	01.01.19	
	31 1	01.01.19	
	32 1	01.01.19	
	28 2	01.01.19	
	29 2	01.01.19	
	31 3	01.01.19	
	32 3	01.01.19	
	30 4	01.01.19	
	31 4	01.01.19	
	31 5	01.01.19	
	32 5	01.01.19	
	30 6	01.01.19	
	31 6	01.01.19	
	31 7	01.01.19	
	32 7	01.01.19	
	31 8	01.01.19	
	32 8	01.01.19	
	30 9	01.01.19	
	31 9	01.01.19	
	31 10	01.01.19	
	32 10	01.01.19	
	30 11	01.01.19	
	31 11	01.01.19	
	31 12	01.01.19	
	32 12	01.01.19	
	1 13	01.01.19	

Perspektiv på afprøvning

- Afprøvning kan forbedre vores tillid til programmet, men kan aldrig bevise at programmet ikke indeholder fejl!
- *Afprøveren* synes at afprøvningen er vellykket hvis den finder fejl.
- *Programmøren* synes at afprøvningen er vellykket hvis den *ikke* finder fejl.
- Når afprøveren og programmør er den samme, er der en psykologisk konflikt.
 - Det tager tid at opstille en afprøvning. Det motiverer til
 - at undgå overflødige valg- og løkke-ordre (forenkler den interne afprøvning);
 - at undgå overflødige specialtilfælde i problemformuleringen (forenkler den eksterne afprøvning).
- Programmer bør afprøves
 - hvis de kan forårsage skade på mennesker eller dyr;
 - hvis de kan forårsage betydelige økonomiske tab;
 - hvis de bruges til at drage videnskabelige konklusioner.
- Nogle programmer behøver man ikke afprøve.