

# Felhantering

## Ofta antar vi att allt ska fungera...

Alla **filer** vi behöver finns går att öppna  
Tillräckligt mycket **minne** finns  
**Serverar** som vi kontaktar svarar  
...

## ...och glömmer att ta hand om fel!

Filer saknas  
Minnet tar slut  
Serverar är inte tillgängliga  
...



**Att upptäcka och signalera fel**

## ■ Exempelproblem: Konvertera *sträng* till *heltal*

### ■ Sträng: "4711" (4 tecken)

- Första siffran är '4' → 4 = 4
- Andra siffran är '7' →  $4 \cdot 10 + 7$  = 47
- Tredje siffran är '1' →  $47 \cdot 10 + 1$  = 471
- Fjärde siffran är '1' →  $471 \cdot 10 + 1$  = 4711
- → 4711, ett *heltal*

### ■ Hur går vi från tecken '4' till heltal 4?

- Java använder **Unicode**
- '4' har numeriskt värde 52
- '0' har numeriskt värde 48
- '4' - '0' =  
52 - 48 =  
4

4 steg

| Dec | Hex | Oct | HTML   | Chr   |
|-----|-----|-----|--------|-------|
| 32  | 20  | 040 | &#032; | Space |
| 33  | 21  | 041 | &#033; | !     |
| 34  | 22  | 042 | &#034; | "     |
| 35  | 23  | 043 | &#035; | #     |
| 36  | 24  | 044 | &#036; | \$    |
| 37  | 25  | 045 | &#037; | %     |
| 38  | 26  | 046 | &#038; | &     |
| 39  | 27  | 047 | &#039; | '     |
| 40  | 28  | 050 | &#040; | (     |
| 41  | 29  | 051 | &#041; | )     |
| 42  | 2A  | 052 | &#042; | *     |
| 43  | 2B  | 053 | &#043; | +     |
| 44  | 2C  | 054 | &#044; | ,     |
| 45  | 2D  | 055 | &#045; | -     |
| 46  | 2E  | 056 | &#046; | .     |
| 47  | 2F  | 057 | &#047; | /     |
| 48  | 30  | 060 | &#048; | 0     |
| 49  | 31  | 061 | &#049; | 1     |
| 50  | 32  | 062 | &#050; | 2     |
| 51  | 33  | 063 | &#051; | 3     |
| 52  | 34  | 064 | &#052; | 4     |
| 53  | 35  | 065 | &#053; | 5     |
| 54  | 36  | 066 | &#054; | 6     |
| 55  | 37  | 067 | &#055; | 7     |
| 56  | 38  | 070 | &#056; | 8     |
| 57  | 39  | 071 | &#057; | 9     |



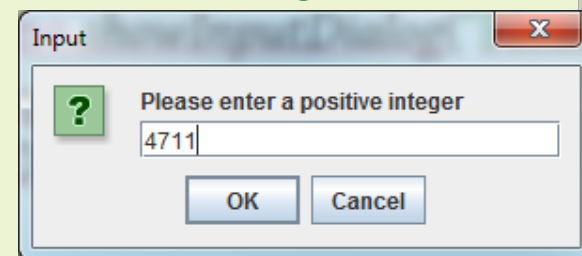
## ■ Kodexempel

```
public class NumberParser {  
    public static int parsePositiveInteger(String str) {  
        int result = 0;  
  
        // För varje tecken  
        for (int pos = 0; pos < str.length(); pos++) {  
            char character = str.charAt(pos);  
  
            // Exempel: '4' - '0' == 52 - 48 == 4  
            int digit = character - '0';  
  
            // Flytta det gamla 1 steg åt vänster; infoga nya siffran  
            result = result * 10 + digit;  
        }  
        return result;  
    }  
}
```

- **Utökning:** Läs in ett positivt heltal från användaren
  - **JOptionPane** ger alltid strängar, tecken för tecken

```
public class NumberParser {  
    public static void main(String[] args) {  
        String answer = JOptionPane.showInputDialog("Please enter a positive integer");  
        int value = parsePositiveInteger(answer);  
        JOptionPane.showMessageDialog(null, "You said: " + value);  
    }  
}
```

```
private static int parsePositiveInteger(String str) {  
    int result = 0;  
    for (int pos = 0; pos < str.length(); pos++) {  
        int digit = str.charAt(pos) - '0';  
        result = result * 10 + digit;  
    }  
    return result;  
}
```

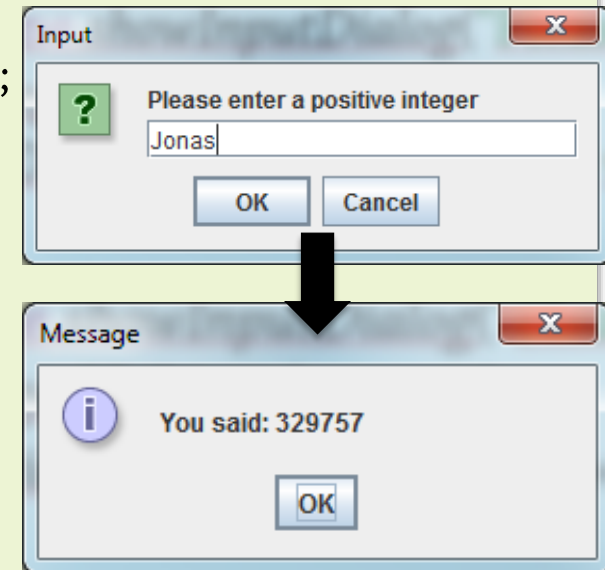


**Var alltid  
misstänksam mot  
användarens indata!**

- Ibland kan användaren göra fel i *inmatningen*

```
public class NumberParser {  
    public static void main(String[] args) {  
        String answer = JOptionPane.showInputDialog("Please enter a positive integer");  
        int value = parsePositiveInteger(answer);  
        JOptionPane.showMessageDialog(null, "You said: " + value);  
    }  
}
```

```
private static int parsePositiveInteger(String str) {  
    int result = 0;  
    for (int pos = 0; pos < str.length(); pos++) {  
        int digit = str.charAt(pos) - '0';  
        result = result * 10 + digit;  
    }  
    return result;  
}
```



# Upptäcka fel: Själv (2)

- "Jonas"

- 'J' - '0' = 74 - 48 = 26

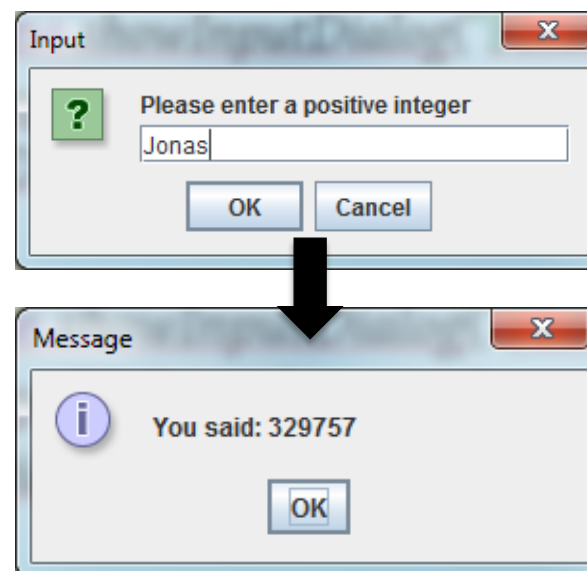
- 'o' - '0' = 111 - 48 = 63

- ➔  $26 * 10 + 63 = 323$

- ...

|       |            |   |
|-------|------------|---|
| 73 49 | 111 &#073; | I |
| 74 4A | 112 &#074; | J |
| 75 4B | 113 &#075; | K |

|        |            |   |
|--------|------------|---|
| 110 6E | 156 &#110; | n |
| 111 6F | 157 &#111; | o |
| 112 70 | 160 &#112; | p |



# Upptäcka fel: Själv (3)

- Ingen kan "upptäcka det åt oss" – det är vi som måste hitta felet

```
private static int parsePositiveInteger(String str) {  
    int result = 0;  
  
    for (int pos = 0; pos < str.length(); pos++) {  
        int digit = str.charAt(pos) - '0';  
  
        if (digit < 0 || digit > 9) {  
            Felaktig input!  
            Vad ska vi göra?  
        }  
  
        result = result * 10 + digit;  
    }  
    return result;  
}
```

Ignorera? Nej...



Laga felet själva? Går inte!

Signalera  
till anroparen!

# Hur signalerar man fel?

- Ett sätt att signalera: Speciella returvärden

```
private static int parsePositiveInteger(String str) {  
    int result = 0;  
  
    for (int pos = 0; pos < str.length(); pos++) {  
        int digit = str.charAt(pos) - '0';  
  
        if (digit < 0 || digit > 9) {  
            return -1;  
        }  
  
        result = result * 10 + digit;  
    }  
    return result;  
}
```

**Metoden läser positiva heltal**  
→ -1 kan aldrig returneras om vi "lyckas"  
→ Kan användas som speciell "signal"

## Returvärden används t.ex. i C

|          |                                                                 |
|----------|-----------------------------------------------------------------|
| fopen(): | NULL → fel                                                      |
| fread(): | 0 → fel                                                         |
| bind():  | 0 → OK, -1 → fel                                                |
| fgetc(): | EOF → filslut <u>eller</u> fel<br>(testa med ferror() / feof()) |

- Nu måste anroparen upptäcka att något gick fel
  - Det signaleras – men man måste titta på signalen

```
public static void main(String[] args) {  
    String answer = JOptionPane.showInputDialog("Please enter a positive integer");  
    int value = parsePositiveInteger(answer);  
    JOptionPane.showMessageDialog(null, "You said: " + value);  
}
```



```
public static void main(String[] args) {  
    int value = -1;  
    while (value < 0) {  
        String answer = JOptionPane.showInputDialog("Please enter a positive integer");  
        value = parsePositiveInteger(answer);  
    }  
    JOptionPane.showMessageDialog(null, "You said: " + value);  
}
```

# Kontrollera returvärde (2)



- Signalering med returvärden kan ge problem...

## 1) Tänk om vi glömmer testa!

```
public static void main(String[] args) {  
    String answer = JOptionPane.showInputDialog("How many people?");  
    int num = parsePositiveInteger(answer);  
    String[] names = new String[num];  
}
```

Krasch:Array med negativ storlek

2) I `parseAnyInteger()` (även negativa)  
finns inget "ledigt" returvärde  
som kan betyda "fel"...



# Kontrollera returvärde (3)

## 3) Jobbigt om det finns mellannivåer

```
row = query.getNextRow()  
if row.error:  
    return error  
else:  
    ...
```

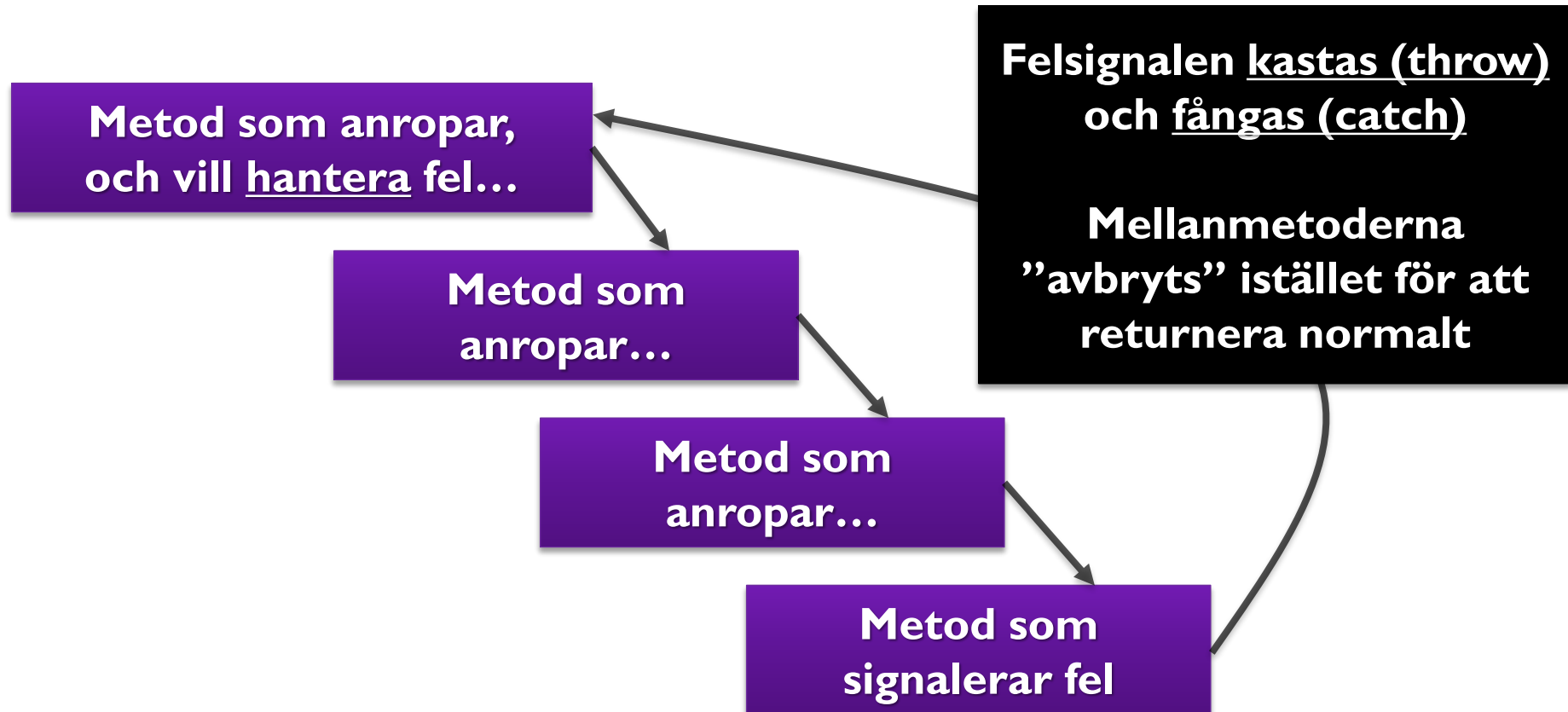
```
def getNextRow():  
    row = driver.get()  
    if row.error:  
        return error  
    else:  
        ...
```

```
def get():  
    data = network.receive()  
    if data.error:  
        return error  
    else:  
        ...
```

```
def receive():  
    if socket.closed():  
        return error  
    else:  
        ...
```

**Exceptions / undantag: Grunderna**

- Ett alternativ:
  1. Inför *speciella kontrollstrukturer* för att signalera fel
  2. Inför *speciella kontrollstrukturer* för att visa var felen kan *hanteras*
  3. Låt felen *automatiskt skickas vidare* ända till hanteraren



# Undantag 2: Signalera fel

- Fel kallas **undantag** (exceptions)

```
private static int parsePositiveInteger(String str) throws NumberFormatException {  
    int result = 0;  
  
    for (int pos = 0; pos < str.length(); pos++) {  
        int digit = str.charAt(pos) - '0';  
  
        if (digit < 0 || digit > 9) {  
            throw new NumberFormatException();  
        }  
  
        result = result * 10 + digit;  
    }  
    return result;  
}
```

Fel hanteras separat →  
Reservera inte returvärdet,  
deklarera feltyper istället

Fel → **kasta** ett undantag  
(avbryter exekveringen,  
inget värde returneras)

# Undantag 3: Att läsa kod – signalering

```
if (digit < 0 || digit > 9) {  
    return -1;  
}
```



Speciell syntax →  
lättare att se var fel signaleras

```
private static int parsePositiveInteger(String str) throws NumberFormatException {  
    int result = 0;  
  
    for (int pos = 0; pos < str.length(); pos++) {  
        int digit = str.charAt(pos) - '0';  
  
        if (digit < 0 || digit > 9) {  
            throw new NumberFormatException();  
        }  
  
        result = result * 10 + digit;  
    }  
    return result;  
}
```

# Undantag 4: Att läsa kod – felhantering

```
String answer = JOptionPane.showInputDialog("Please enter a positive integer");  
int value = parsePositiveInteger(answer);  
if (value >= 0) {  
    String[] names = new String[num];  
    // ... mer kod som körs om allt är OK ...  
} else {  
    // Fel format  
}
```

Gammal felhanterare: Vanlig villkorssats mitt i en funktion



```
String answer = JOptionPane.showInputDialog("Please enter a positive integer");  
int value = 0;  
try {  
    value = parsePositiveInteger(answer);  
    String[] names = new String[num];  
    // ... mer kod som körs om allt är OK ...  
} catch (NumberFormatException e) {  
    // Fel format  
}
```

Exceptions: Egen kontrollstruktur, try / catch  
Felhanteringskod separeras från övrig kod

# Undantag 5: Skicka vidare, automatiskt



```
public static void main(String[] args) {  
    try {  
        int value = getNumberFromUser();  
        JOptionPane.showMessageDialog(null, "You said: " + value);  
    } catch (NumberFormatException e) {  
        // ...  
    }  
}
```

Kan automatiseras för att vi  
vet vad som är en felsignal

Fångas och hanteras här

```
private static int getNumberFromUser() throws NumberFormatException {  
    String answer = JOptionPane.showInputDialog("Please enter a positive integer");  
    return parsePositiveInteger(answer);  
}
```

Fångas inte här → skickas vidare

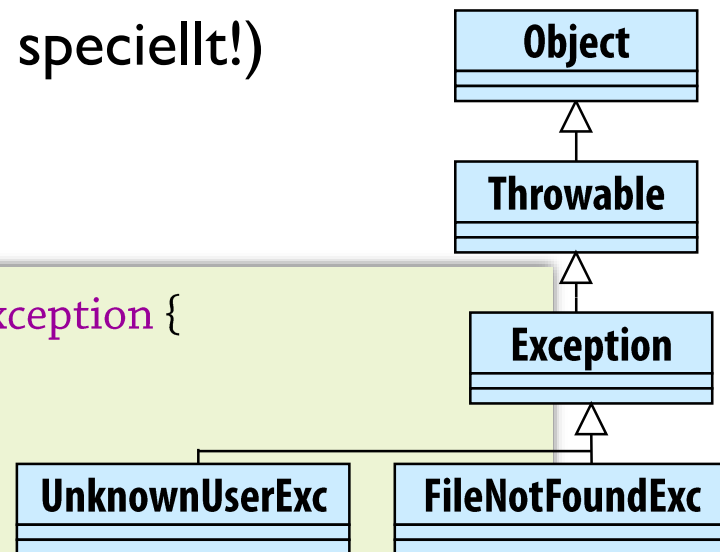
```
private static int parsePositiveInteger(String str) throws NumberFormatException {  
    int result = 0;  
    for (int pos = 0; pos < str.length(); pos++) {  
        int digit = str.charAt(pos) - '0';  
        if (digit < 0 || digit > 9) throw new NumberFormatException();  
        result = result * 10 + digit;  
    }  
    return result;  
}
```

Fångas inte här:  
Ligger inte inom try/catch  
→ Går automatiskt till anroparen

# Undantag 6: Objekt

- I Java är undantag **objekt** (som hanteras speciellt!)
  - Typ: Subklass till **Throwable**

```
public class FileNotFoundException extends Exception {  
    public FileNotFoundException(String msg) {  
        super(msg);  
    }  
}
```



```
if (...) throw new FileNotFoundException(filename);  
// Skapa ett exception-objekt, och kasta det sedan (signalera fel)
```

```
public class UnknownUserException extends Exception {  
    private String name;           // Name of unknown user  
    private String database;       // Database used for lookup  
    public UnknownUserException(String name, String database) { ... }  
}
```

Har fält, konstruktorer  
Kan lagra felinformation



# Undantag och kontrakt

- Metoder **deklarerar** vilka undantag som kan kastas

```
public class FileInputStream extends InputStream
{
    public FileInputStream(String name) throws FileNotFoundException {
        ...;
    }
}
```

**Ingår i kontraktet!**

**"Jag lovar att inte signalera några andra fel än FileNotFoundException"**

**Behöver inte titta i kod eller dokumentation för att se vilka fel som kan uppstå, hur de signaleras**

|          |                                |
|----------|--------------------------------|
| fopen(): | NULL → fel                     |
| fread(): | 0 → fel                        |
| bind():  | 0 → OK, -1 → fel               |
| fgetc(): | EOF → filslut <u>eller</u> fel |

# Kontrakt 2: Underklasser

- Vi vet: Kontraktslöften får inte försvagas i underklasser!

```
public class StreamReader
{
    public int readInt() throws IOException {
        ...;
    }
}
```

```
public class MyReader extends StreamReader
{
    @Override
    public int readInt() throws IOException, OverflowException {
        ...;
    }
}
```

**Kompilatorn klagar: StreamReader säger  
att bara IOException kan kastas**

- Kompilatorn kontrollerar att kontraktet efterlevs

## Måste fånga med try-catch:

```
public String readFile(String name) {  
    try {  
        InputStream in = new FileInputStream(name); // Kan kasta FNFE  
        // ...  
    } catch (FileNotFoundException ex) {  
        ...  
    }  
}
```

## Eller tala om att fel kan skickas vidare:

```
public String readFile(String name) throws FileNotFoundException {  
    InputStream in = new FileInputStream(name);  
    // ...  
}
```

Med speciella "felvärden"  
kunde vi glömma att testa returvärdet...

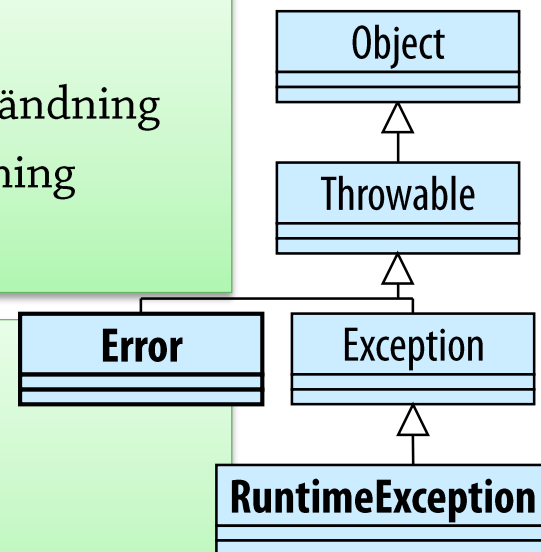
# Kontrakt 4: Undantag (!)

**RuntimeException** kan inträffa nästan var som helst

- **ArrayIndexOutOfBoundsException** – varje arrayanvändning
- **NullPointerException** – varje pekaranvändning
- **ClassCastException** – varje cast

**Error**: Vissa allvarliga fel

- **OutOfMemoryError** – varje **new**
- **StackOverflowError** – varje metदानrop
- **ClassFormatError** – varje klass som används



- Ska fortfarande **representeras som undantag**, inte "krascha"
  - Lättare att felsöka
- Ska normalt **inte fångas**
  - Interna datastrukturer kan ha "förstörts",  
just eftersom man inte kan gardera sig mot det som kan inträffa var som helst
- Behöver inte deklarereras – ingår ej i kontrakt

# Hantera undantag: När och hur?

Ibland kan man uppfylla kontraktet trots felet

```
/** Visualizes in 3D or 2D depending on  
current graphics capabilities */
```

```
void visualize() {  
    try {  
        visualize3D();  
    } catch (No3DGraphicsException e) {  
        visualize2D();  
    }  
}
```

```
/** Signal task completion visually,  
and possibly audibly */
```

```
void signalCompletion() {  
    statusBar.setText("Done!");  
    try {  
        beep();  
    } catch (NoSoundException e) {  
        // Not possible, but we have still  
        // satisfied our contract...  
    }  
}
```

På lägre nivå uppstod ett fel  
På högre nivå var detta *inte viktigt*, målet nåddes ändå

Måste inte informera någon – men ibland vill man ändå logga felet

# Hantera 2: Skicka vidare

Kan man inte uppfylla kontraktet måste anroparen informeras!

```
public class DocumentIO {  
    /** Saves the file to disk */  
    void saveToFile(Document doc) throws IOException {  
        // Försök spara filen.  
        FileOutputStream fs = new FileOutputStream();  
        ...  
        ...  
    }  
}
```

Informera genom att inte fånga fel:  
**IOException** från **FileOutputStream**  
skickas automatiskt vidare



Man kan alltid tillfälligt fånga upp felet

```
public class DocumentIO {  
    public void saveToFile(Document doc) throws IOException {  
        try {  
            // save the file ...  
        }  
        catch (IOException e) {  
            // Some cleanup: Delete temp file...  
            throw e;  
        }  
    }  
}
```

Fånga upp felet,  
städa,  
kasta vidare *samma* objekt

# Hantera 4: Gemensamt eller separat?



## Gemensamt try/catch-block

```
void single() {  
    final InputStream is1, is2;  
  
    try {  
        is1 = new FileInputStream("Hello");  
        is2 = new FileInputStream("Goodbye");  
    } catch (FileNotFoundException e) {  
        // Executed if either file is missing  
    }  
}
```

Mindre kod, lättare att läsa

## Separata try/catch-block

```
void multiple() {  
    final InputStream is1, is2;  
  
    try {  
        is1 = new FileInputStream("Hello");  
    } catch (FileNotFoundException e) {  
        // Executed if Hello is missing  
    }  
    try {  
        is2 = new FileInputStream("Goodbye");  
    } catch (FileNotFoundException e) {  
        // Executed if Goodbye is missing  
    }  
}
```

Vet på vilken rad felet uppstod

Anpassa till situationen!

# Hantera 5: Vad ska jag fånga?

“Catch” fångar fel av en typ **och dess subtyper**

```
public class ExecuteExternal {  
    public static void execute(final String[] args) {  
        try {  
            final Process p = Runtime.getRuntime().exec(args);  
            p.waitFor();  
        } catch (Exception e) {  
            // Executed if anything  
            // goes wrong  
        }  
    }  
}
```

Fångar alla feltyper på en plats:  
Ser inte *vilka* fel som kan uppstå  
Hur vet man om varje feltyp hanteras rätt?

"Exception" är *definitivt* för brett:  
Fångar även RuntimeExceptions (null-pekare, ...)

```
try {  
    ...  
} catch (IOException e) {  
    ...  
} catch (InterruptedException e) {  
    ...  
}
```

Ange specifika fel  
som ska hanteras

```
try {  
    ...  
} catch (IOException | InterruptedException e) {  
    ...  
}
```

Om man vill hantera flera  
feltyper på samma sätt

# Felfel 1: Fånga och ignorera → följdfel

```
public class WordProcessor {  
    private Document doc = null;  
  
    public WordProcessor(String filename) {  
        loadFile(filename);  
        System.out.println("Size is " + doc.getLength());  
    }  
  
    private void loadFile(String filename) {  
        try {  
            FileInputStream is = new FileInputStream(filename);  
            // ...  
            doc = parseDocumentFrom(is);  
        } catch (FileNotFoundException e) {  
            e.printStackTrace(); // Felmeddelande  
        }  
    }  
}
```

Tror allt gick bra,  
men doc är fortfarande null  
→ **NullPointerException**

I många projekt inträffar  
kraschen långt senare  
→ svårt att hitta ursprungsfelet

Vanligt felhanteringsfel:  
Skriver ut felmeddelande, informerar *inte* anroparen

# Felfel 1b: Fånga och ignorera → följdfelet

```
public class WordProcessor {  
    private Document doc = null;  
  
    public WordProcessor(String filename) {  
        try {  
            loadFile(filename);  
        } catch (FileNotFoundException e) {  
            System.out.println("File not found");  
        }  
        System.out.println("Size is " + doc.getLength());  
    }  
  
    private void loadFile(String filename) throws FileNotFoundException {  
        FileInputStream is = new FileInputStream(filename);  
        // ...  
        doc = parseDocumentFrom(is);  
    }  
}
```

...men anroparen själv fångar felet,  
ignorerar det,  
fortsätter,  
och kraschar senare!

Här är loadFile() korrekt...

# Undvik felhanteringsfel!

- För att undvika problem, tänk alltid:
  - Vad händer *efter* jag fångar upp felet?
  - Uppfyller jag mina löften?
  - Får någon *reda* på om jag inte uppfyller mina löften?
  - *Hur fortsätter exekveringen av programmet?*

# Vem informerar användaren?

- Vi delar upp klasser i två kategorier

**De flesta klasser och metoder används bara av *andra klasser***

**Misslyckanden ska signaleras till anroparen**

**Kan också loggas till loggfil**

**Vissa klasser och metoder tillhör användargränssnittet**

**Får prata med användaren**

**Textutmatning,  
GUI**

# Informera 2: Inte lågnivåklasser

Om användaren måste informeras:

```
public class DocumentIO {  
    public void saveToFile(Document doc) throws IOException {  
        try {  
            // save the file ...  
        }  
        catch (IOException e) {  
            // Some cleanup: Delete temp file...  
            System.out.println("Couldn't save!");  
            throw e;  
        }  
    }  
}
```

**Inte av en “lågnivåklass”!**

Inte ens om klassen också  
informerar anroparen, som här

Denna klass är ett verktyg,  
inte del av användargränssnittet  
(som kanske föredrar *dialogruta*)!



Klasser i **användargränssnittet**  
kan **själva** informera användaren

```
public class WordProcessorUI {  
  
    /** Try to save the file. If impossible,  
        inform the user through a dialog. */  
    void saveOrInform() {  
        try {  
            this.docIO.saveToFile(this.doc);  
        } catch (IOException e) {  
            JOptionPane.showMessageDialog(...);  
        }  
    }  
}
```

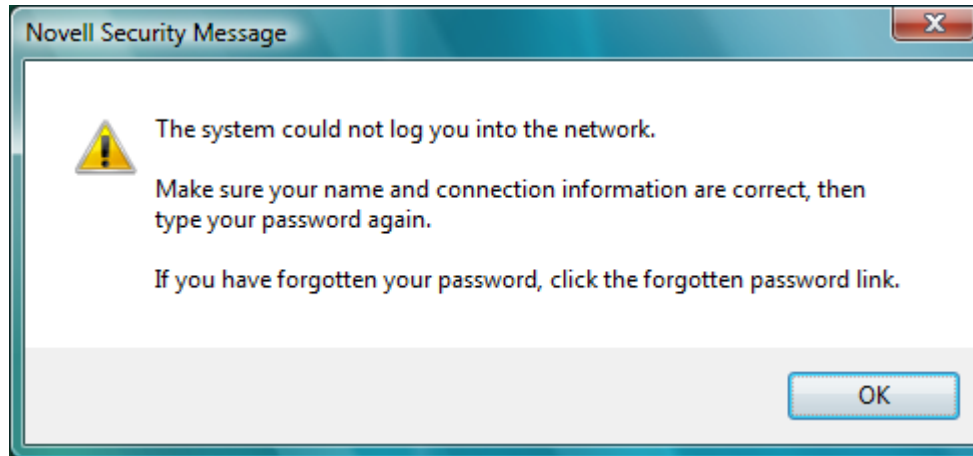
**Kontraktet anpassas:  
saveOrInform säger inte  
”sparar definitivt filen”.**

# Felfel 2: Fånga och gissa

```
try {  
    connectToDatabase();  
} catch (DatabaseException e) {  
    System.out.println("Wrong password!");  
    System.exit(1);  
}
```

Glöm att starta databasen →  
spenderar tre timmar på att hitta rätt lösenord

- Ett "välpolerat" program visar (helst) inte "råa" felmeddelanden
  - Anpassar dem till användaren



# Informera 5: På låg nivå



- Som utvecklare behöver vi mer detaljer!
  - Finns i undantaget vi fångar – men skriv inte bara ut det!
    - **System.out.println(e);**
    - Säger inget om var felet inträffade
  - Skriv *åtminstone* ut en *stack trace*
    - **e.printStackTrace();**
    - Exception in thread "main" java.lang.NullPointerException  
at com.example.myproject.Book.getTitle(Book.java:16)  
at com.example.myproject.Author.getBookTitles(Author.java:25)  
at com.example.myproject.Bootstrap.main(Bootstrap.java:14)
  - Om felet absolut inte kan inträffa (“when pigs fly”):
    - Det kan det nog ändå
    - Skriv *åtminstone* ut en *stack trace*
  - Eller logga felet till fil!



the impossible happens.®

**Undantag i undantagsfall**

# Helt fel: Använda undantag i normalfall

```
public class Calculator {  
    public static int sum(int[] array) {  
        int sum = 0;  
        int pos = 0;  
        try {  
            while (true) {  
                sum = sum + array[pos];  
                pos++;  
            }  
        } catch (ArrayIndexOutOfBoundsException e) {  
            // OK, we're at the end of the array  
        }  
        return sum;  
    }  
}
```

Undantag ska användas i undantagsfall

Här blir det en exception vid *varje* anrop!

**Kasta undantag:  
Vilken typ?**

# Typer 1: Inbyggda undantagstyper

- Undantagsklasser ska vara så specifika som möjligt, för att:
  - Se exakt vilka problem som kan inträffa
  - Fånga specifika fel, utan att fånga andra
  - Ge förståeliga felmeddelanden
- Använd inbyggda feltyper där de passar!
  - Exempel: `IndexOutOfBoundsException`

```
public class ArrayList {  
    private int size;  
    ...  
    private Object get(int index) {  
        if (index < 0 || index > size) {  
            throw new IndexOutOfBoundsException  
                ("Index " + index + " not in bounds [0," + (size-1 + ")]");  
        }  
        ...  
    } }  
}
```

Passar perfekt!



- Om det inte finns några passande typer?

- Gör inte så här:

- **throw new** `IOException`("Database error")
- **throw new** `IOException`("User not found")
- **throw new** `IOException`("User already exists")

**Kan inte särskiljas av programmet!**

Anroparen kan inte skilja "user already exists" från "database error", osv.

- Skapa en egen undantagstyp!

```
public class NoSuchUserException extends Exception {  
    public NoSuchUserException(String message) {  
        // Pass the error message to the superclass constructor  
        // (will be shown in the stack trace)  
        super(message);  
    }  
}
```

**Assertions:**

**Att påstå, försäkra, bedyra att något är sant**

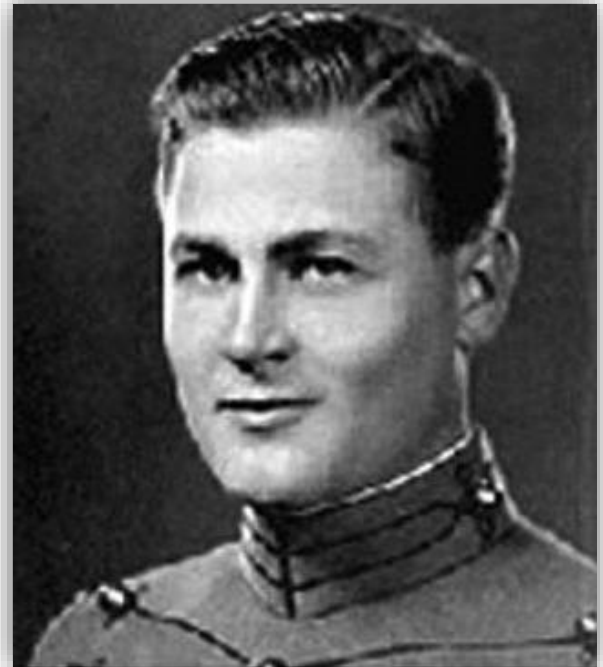
- Hur upptäcker vi alla fel som kan uppstå?

## Murphys lag

Framställs ofta som  
"Om något kan gå fel  
kommer det att göra det"

### Gäller även här:

Egentligen sa han att  
*"Om det finns två eller fler sätt att göra  
något, och ett av dessa sätt leder till en  
katastrof, kommer någon att göra det  
på det sättet" –*  
**förespråkade defensiv design**



*Edward Aloysius Murphy, Jr.*

Ingenjör i amerikanska  
flygvapnet

Ett steg på vägen:

Var alltid misstänksam

Testa att antaganden verkligen stämmer

```
class PriorityQueue {  
    private Comparable[] elements;  
    private int size;  
  
    public void insert(Comparable element) {  
        // Very complex code: Insert element,  
        // rebalance structure for fast access!  
    }  
}
```

Vi är säkra på att  
det här fungerar!

Men om det *finns*  
ett fel, kan det bli  
våldigt svårt att  
upptäcka  
(inträffar sällan)

# Assertions 2: Var misstänksam



```
class PriorityQueue {  
    private Comparable[] elements;  
    private int size;  
  
    public void insert(Comparable element) {  
        // Very complex code: Insert element,  
        // rebalance structure for fast access!  
        ...  
        // Did this work correctly? Let's check!  
        if (!isSorted(elements))  
            throw new BugException("Internal error: Queue not sorted");  
    }  
    private boolean isSorted(Comparable[] array) {  
        // This test is simple -- hard to get it wrong!  
        for (int i = 0; i+1 < array.length; i++) {  
            if (array[i].compareTo(array[i+1]) > 0) return false;  
        }  
        return true;  
    }  
}
```

Vi är defensiva och misstänksamma:  
Stoppar in ett test

Tar för mycket tid i slutgiltiga versionen!  
Vill ha en debugflagga för att slå av och på tester...

- För att få in värdet på en flagga:
  - **Kommandoradsparametrar:** Hantera själv

```
java mypackage.MyProgram -verify thefile.doc
```

```
public class MyProgram {  
    public static void main(String[] args) {  
        // Kolla i args  
    }  
}
```

Bra för "vanliga"  
parametrar

- **System properties:** Hanteras av Java, görs tillgängliga "globalt"

```
java -Dverify=true mypackage.MyProgram thefile.doc
```

```
public class PriorityQueue {  
    private final static boolean verify = System.getProperty("verify") != null;  
}
```

Bra för utvecklarens  
kontroll på låg nivå

# Assertions 3: Prestanda

```
class PriorityQueue {  
    private Comparable[] elements;  
    private int size;  
    private final static boolean verify = System.getProperty("verify") != null;
```

```
    public void insert(Comparable element) {  
        // Very complex code: Insert element,  
        // rebalance structure for fast access!
```

```
        ...
```

```
        // Did this work? Let's check!
```

```
        if (verify && !isSorted(elements))
```

```
            throw new BugException("Internal error: Queue not sorted");
```

```
    }
```

```
    private boolean isSorted(Comparable[] array) {
```

```
        // Simple, hard to get it wrong!
```

```
        for (int i = 0; i+1 < array.length; i++) {
```

```
            if (array[i].compareTo(array[i+1]) > 0) return false;
```

```
        }
```

```
        return true;
```

```
    }
```

```
}
```

**Statisk konstant →  
sätts när klassen laddas  
(Java har redan läst kommandoraden)**

Java optimerar kod vid körning

**PriorityQueue.verify == false?**

→ Optimeraren tar bort testet  
– koden kör i full fart!

**Jättebra, men...  
Kan vi förenkla det?**



# Assertions 4: Inbyggd nyckelord

```
class PriorityQueue {  
    private Comparable[] elements;  
    private int size;  
  
    public void insert(Comparable element) {  
        // Very complex code: Insert element,  
        // rebalance structure for fast access!  
  
        ...  
        // Did this work? Let's check!  
        assert isSorted(elements) : "Internal error: Queue not sorted";  
    }  
    private boolean isSorted(Comparable[] array) {  
        // Simple, hard to get it wrong!  
        for (int i = 0; i+1 < array.length; i++) {  
            if (array[i].compareTo(array[i+1]) > 0) return false;  
        }  
        return true;  
    }  
}
```

**java -enableassertions ...**  
**(short: -ea)**

Kan styras per paket, klass, ...

- Javas **assert**-sats:
  - Talar om vad du "vet" definitivt är sant
    - **assert** condition : errorMessage;
  - Ungefär lika med:
    - **if** (assertionsEnabled && !condition) {  
    **throw new** **AssertionError**(errorMessage);  
}

# Assertions vs. Undantag

- Använd assertions för att fånga interna buggar
  - ...men undantag för att signalera fel från anroparen!

Bara falskt  
om klassen  
har ett fel  
→  
Kan använda  
assertions

```
class PriorityQueue {  
    private Comparable[] elements;  
    private int size;  
  
    public void insert(Comparable element) {  
        ...  
        → assert isSorted(elements) : "Internal error: Queue not sorted";  
    }  
    /** @throws NoSuchElementException if the queue is empty */  
    public Object pop() {  
        → if (size == 0) throw new NoSuchElementException();  
        else return elements[size--];  
    }  
}
```

Kolla fel  
argument  
från extern  
anropare →  
Undantag

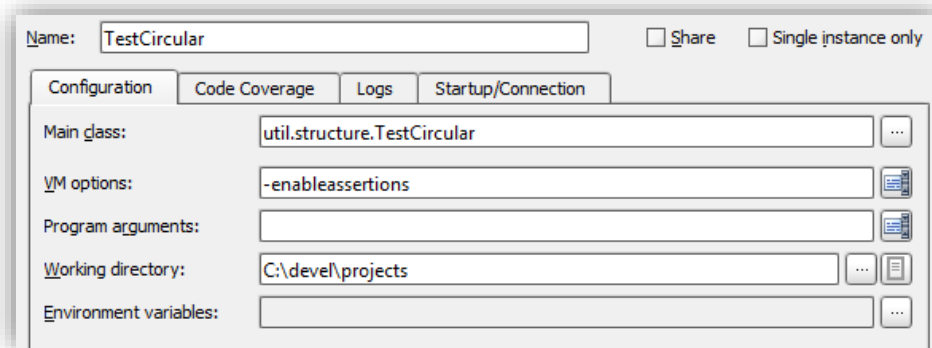
En assertion här skulle betyda "jag vet att ingen någonsin kör pop() när kön är tom" – men det beror på anroparen

- Kom ihåg: Påstå bara det som du "vet" är sant

- Assertions kontrolleras **bara** om parameter `-ea` anges!

- Annars optimeras den bort vid körning

- Behåll assertions i produktionskod, be **användarna** slå på dem om de har problem



- ➔ Ännu viktigare att bara använda dem för att verifiera det man “vet”!

```
public void doSomethingDangerous() {  
    assert hasPrivileges() : "Sorry, you can't do that!";  
    // FEL: Utan -ea sker ingen kontroll... Använd if + undantag!  
}
```

```
/** @param foo Must not be null */  
public void addToImportantSystemTable(Object foo) {  
    assert foo != null;  
    table.add(foo); // FEL: Förstör interna strukturer om foo == null  
}
```

**Att logga fel (och annat)**

- Många program **loggar** vad som händer
  - Uppstart och parametrar – underlättar senare felsökning

```
2015-02-03 09:17:21,476 [ 0] INFO - ----- IDE STARTED -----  
-----  
2015-02-03 09:17:21,520 [ 44] INFO - IDE:IntelliJ IDEA (build #IU-139.463.4, 19 Nov 2014 00:00)  
2015-02-03 09:17:21,520 [ 44] INFO - OS:Windows 7 (6.1,x86)  
2015-02-03 09:17:21,520 [ 44] INFO - JRE: 1.8.0_25-b18 (Oracle Corporation)  
2015-02-03 09:17:21,520 [ 44] INFO - JVM: 25.25-b02 (Java HotSpot(TM) Server VM)  
2015-02-03 09:17:21,528 [ 52] INFO - JVM Args: -Xms128m -Xmx512m -XX:MaxPermSize=250m -  
XX:ReservedCodeCacheSize=150m -ea -Dsun.io.useCanonCaches=false -Djava.net.preferIPv4Stack=true -  
Djsse.enableSNIExtension=false -XX:+UseConcMarkSweepGC -XX:SoftRefLRUPolicyMSPerMB=50 -  
XX:+HeapDumpOnOutOfMemoryError -Xbootclasspath/a:C:\Program Files (x86)\JetBrains\IntelliJ IDEA  
14.0\lib\boot.jar -Didea.paths.selector=IntelliJ14  
2015-02-03 09:17:21,899 [ 423] INFO - JNA library loaded (32-bit) in 371 ms  
2015-02-03 09:17:21,918 [ 442] INFO - penapi.util.io.win32.IdeaWin32 - Native filesystem for Windows  
is operational  
2015-02-03 09:17:21,934 [ 458] INFO - Using "FocusKiller" library to prevent focus stealing.
```

- Under programkörning – vad som sker, hur lång tid det tar, ...

```
2015-02-03 09:17:37,687 [ 16211] INFO - unicator.p2p.UserMonitorThread - Force finding users
2015-02-03 09:17:37,687 [ 16211] INFO - unicator.p2p.UserMonitorThread - Start User Monitor Thread
2015-02-03 09:17:37,688 [ 16212] INFO - icator.p2p.MulticastPingThread - /192.168.56.1 IDEtalk Multicast Thread:
Start thread.
2015-02-03 09:17:37,689 [ 16213] INFO - icator.p2p.MulticastPingThread - /130.236.184.131 IDEtalk Multicast Thread:
Start thread.
2015-02-03 09:17:39,486 [ 18010] INFO - indexing.UnindexedFilesUpdater - Indexable files iterated in 1527 ms
2015-02-03 09:17:39,487 [ 18011] INFO - indexing.UnindexedFilesUpdater - Unindexed files update started: 428 files
to update
2015-02-03 09:17:39,743 [ 18267] INFO - bugs.resources.ResourcesLoader - Loading locale properties for 'en_US)
2015-02-03 09:17:40,215 [ 18739] INFO - rains.ide.PooledThreadExecutor - Not enough pooled threads; dumping
threads into a file
2015-02-03 09:17:42,296 [ 20820] INFO - indexing.UnindexedFilesUpdater - Unindexed files update done in 2809 ms
2015-02-03 09:17:42,429 [ 20953] INFO - unicator.p2p.UserMonitorThread - Force finding users
2015-02-03 09:17:42,739 [ 21263] INFO - CompilerWorkspaceConfiguration - Available processors: 8
2015-02-03 09:17:43,422 [ 21946] INFO - tor.impl.FileEditorManagerImpl - Project opening took 9363 ms
2015-02-03 09:17:44,163 [ 22687] INFO - or.jabber.impl.JabberTransport - Jabber connected
2015-02-03 09:17:53,060 [ 31584] INFO - dvertisement.PluginsAdvertiser - Read timed out
```

- **Fel** som inträffar

```
2015-02-03 09:19:35,574 [ 134098] INFO - ins.idea.svn.SvnChangeProvider - svn: E155007:
'C:\devel\svnwc2\teach\java\source\app\clock\ClockComponent.java' is not a working copy
org.jetbrains.idea.svn.commandLine.SvnBindException: svn: E155007:
```

```
'C:\devel\svnwc2\teach\java\source\app\clock\ClockComponent.java' is not a working copy
```

```
    at org.jetbrains.idea.svn.status.SvnKitStatusClient.doStatus(SvnKitStatusClient.java:82)
    at org.jetbrains.idea.svn.SvnChangeProvider.processCopiedFile(SvnChangeProvider.java:256)
    at org.jetbrains.idea.svn.SvnChangeProvider.processCopiedAndDeleted(SvnChangeProvider.java:169)
    at org.jetbrains.idea.svn.SvnChangeProvider.getChanges(SvnChangeProvider.java:100)
    at com.intellij.openapi.vcs.changes.ChangeListManagerImpl.a(ChangeListManagerImpl.java:738)
    at com.intellij.openapi.vcs.changes.ChangeListManagerImpl.a(ChangeListManagerImpl.java:655)
    at com.intellij.openapi.vcs.changes.ChangeListManagerImpl.d(ChangeListManagerImpl.java:530)
    at com.intellij.openapi.vcs.changes.ChangeListManagerImpl.access$1200(ChangeListManagerImpl.java:75)
    at com.intellij.openapi.vcs.changes.ChangeListManagerImpl$ActualUpdater.run(ChangeListManagerImpl.java:438)
    at com.intellij.openapi.vcs.changes.UpdateRequestsQueue$MyRunnable.run(UpdateRequestsQueue.java:260)
    at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
    at java.util.concurrent.FutureTask.run(FutureTask.java:266)
    at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$201(ScheduledThreadPoolExecutor.java:180)
    at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(ScheduledThreadPoolExecutor.java:293)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
    at java.lang.Thread.run(Thread.java:745)
```

```
Caused by: org.tmatesoft.svn.core.SVNException: svn: E155007:
```

```
'C:\devel\svnwc2\teach\java\source\app\clock\ClockComponent.java' is not a working copy
```

```
    at org.tmatesoft.svn.core.internal.wc.SVNErrorManager.error(SVNErrorManager.java:64)
    at org.tmatesoft.svn.core.internal.wc.SVNErrorManager.error(SVNErrorManager.java:51) ... ..
```



- Java: Inbyggt i paketet java.util.logging

```
public class Visualizer {  
    private final static Logger logger = Logger.getLogger(Visualizer.class.getName());  
  
    public void visualize() {  
        try {  
            visualize3D();  
        } catch (No3DGraphicsException e) {  
            logger.log(Level.WARNING, "No 3D graphics, running 2D visualization instead", e);  
            visualize2D();  
        }  
    }  
    ...  
}
```

Ger fullständigt namn för klassen Visualizer

Ger oss en loggskrivare för den här klassen (klassnamn kan visas i loggfilen)

Loggar ett meddelande på WARNING-nivå tillsammans med undantaget **e**

Nivåer: SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST

Särskilt viktigt för undantag som beror på buggar i programmet

- Flera nivåer – så att man kan **filtrera**
  - Lämna alla logganrop i koden
  - Filtrera med `logger.setLevel(Level.INFO);`

- Var hamnar loggmeddelanden?
  - Default: Skrivs ut på standard error
    - **feb 28, 2025 9:29:00 FM exceptions.Visualizer visualize  
WARNING: No 3D graphics, running 2D visualization instead  
exceptions.No3DGraphicsException: Running in the console  
at exceptions.Visualizer.visualize3D(Visualizer.java:27)  
at exceptions.Visualizer.visualize(Visualizer.java:15)  
at exceptions.Visualizer.main(Visualizer.java:11)**
  - Kan även loggas till fil
    - Kräver konfigurationsfil – exempel: *logconfig.properties* innehåller
      - `handlers = java.util.logging.FileHandler`
      - `java.util.logging.FileHandler.level = WARNING`
      - `java.util.logging.FileHandler.append = true`
      - `java.util.logging.FileHandler.pattern = log.%u.%g.txt`
    - **`java -Djava.util.logging.config.file=logconfig.properties ...`**
  - <http://docs.oracle.com/javase/8/docs/technotes/guides/logging/overview.html>

As far as we know,  
our computer has never had an undetected error.

— Conrad H. Weisert