9. Ausnahmebehandlung

Java-Beispiele:

Ausnahme.java
TryCatch.java
TryCatchAll.java
Finally.java
TryInTry.java
KeyboardTry.java
Oeffnungszeit.java

Schwerpunkte

- Ausnahmen und Laufzeitfehler
- Stack-Trace
- Java-Ausnahmeklassen-Hierarchie
- try-catch:
  Nutzerdefinierte Ausnahmebehandlung
- trow:
  Nutzerdefinierte Erzeugung von Ausnahmen

**Allgemeines Konzept in jeder Programmiersprache;
in Java mit objektorientierte Lösung (Ausnahmen sind Objekte)**

Ausnahmen

- Besondere Bedingungen zur Laufzeit des Programms: Fehlersituation
- Keine sinnvolle Weiterarbeit:
  Laufzeitfehler → Abbruch des Programms

Beispiele:

- Array: falscher Index a[i]
- Eingabe: Zahl erwartet
  - aber Buchstabe liegt an
- Zugriff auf Objekt (Aufruf von Methoden):
  - existiert aber nur 'null'-Objekt
- Klasse zur Laufzeit nicht vorhanden
  (nicht compiliert oder nach Compilation: Klasse gelöscht)

Ziel:

Trotz Ausnahme soll sinnvolle Weiterarbeit möglich sein

Laufzeitfehler und Ausnahmeinformationen

Stack-Trace

Beispiele: Laufzeitfehler erzeugen Ausnahmen

```
class Ausnahme   {
   public static void main (String[] args) {
       int i = Integer.parseInt(args[0]);
       System.out.println("i = " + i);
   }
}
```

```
% java Ausnahme 3
```

Ausgabe?

Wirkung von parseInt():

```
Integer.parseInt("125") → 125
```


Beispiele: Laufzeitfehler erzeugen Ausnahmen

```
class Ausnahme   {
   public static void main (String[] args) {
       int i = Integer.parseInt(args[0]);
       System.out.println("i = " + i);
   }
}
```

Ausgabe?

```
% java Ausnahme 3
i = 3

% java Ausnahme
java.lang.ArrayIndexOutOfBoundsException: 0
   at Ausnahme.main

% java Ausnahme a1
java.lang.NumberFormatException: a1
   at java.lang.Integer.parseInt
   at Ausnahme.main
```

Vergleiche: Verschiedene 'at'-Angaben

Ausgabe für die Folie leicht vereinfacht

Inhalt von Ausnahme-Mitteilungen

```
class Ausnahme   {
   public static void main (String[] args){
       int i = Integer.parseInt(args[0]);
       System.out.println("i = " + i);
   }
}
```

```
% java Ausnahme a1
java.lang.NumberFormatException:      a1
   at java.lang.Integer.parseInt
   at Ausnahme.main
```

Was?

Art

Details

Wo: In welcher Umgebung? → Stack-Trace

Wo: In welcher Methode (welcher Klasse)?

Stack-Trace

Ausnahme.java

Umgekehrte Liste aller aktivierten, aber noch nicht beendeten Methoden-Aufrufe

```
class Ausnahme   {
   static int makeIntFromString (String s)   {
       return Integer.parseInt(s);
   }
   public static void main (String[] args)   {
       int i = makeIntFromString(args[0]);
       System.out.println("i = " + i);
   }
}
```

```
% java Ausnahme a1
```

Aufruf 1 main()

Aufruf 2 makeIntFromString("a1")

Aufruf 3 parseInt("a1")

aktivierte Methodenaufrufe im Stack *(LIFO)*

```
parseInt
makeIntFromString
main
```

Stack-Trace

Ausnahme.java

Umgekehrte Liste aller aktivierten, aber noch nicht beendeten Methoden-Aufrufe

```
class Ausnahme   {
   static int makeIntFromString (String s)   {
       return Integer.parseInt(s);
   }
   public static void main (String[] args)   {
       int i = makeIntFromString(args[0]);
       System.out.println("i = " + i);
   }
}
```

```
% java Ausnahme a1
java.lang.NumberFormatException:     a1
   at java.lang.Integer.parseInt          Aufruf 3
   at Ausnahme.makeIntFromString          Aufruf 2
   at Ausnahme.main                       Aufruf 1
```

Ausgabe-richtung (Stack)

```
parseInt
makeIntFromString
main
```


Stack–Trace: Ausschriften nicht einheitlich in Java-Versionen

Java™ Platform Standard Ed. 6

Konstruktor zur Erzeugung von Ausnahmeobjekten → API-Klasse wird durch Java-Interpreter gerufen

```
% java Ausnahme a1
Exception in thread "main" java.lang.NumberFormatException:a1
at java.lang.NumberFormatException(Unknown Source)
at java.lang.Integer.parseInt(Unknown Source)
at Ausnahme.makeIntFromString(Ausnahme.java:6)
at Ausnahme.main(Ausnahme.java:10)
```

„unknown source“: Java API

main() befindet sich im File Ausnahme.java in Zeile 10

Stack-Trace (Teil: Anwenderprogramm)

Ausnahmebehandlung in Java

Bisher:

Nach Ausnahme Abbruch des Programms

Ziel:

Trotz Ausnahme soll sinnvolle Weiterarbeit möglich sein

Ausnahmebehandlung in Java: objektorientierte Lösung

Ausnahme.java

```
class Ausnahme   {
   static int makeIntFromString (String s) {
       return Integer.parseInt(s);
   }
   public static void main (String[] args) {
       int i = makeIntFromString(args[0]);
       System.out.println("i = " + i);
   }
}
```

Aufrufreihenfolge (mit aktuellen Parametern) :

```
% java Ausnahme a1  4  61
```

main({"a1", "4", "61"})

makeIntFromString("a1")

Integer.parseInt("a1")

Java-VM-Maschine:
- entdeckt Ausnahmesituation
- erzeugt Objekt (einer Ausnahmeklasse)

Wie kann man sich Objekte zur Beschreibung von Ausnahmen vorstellen?

Java-VM-Maschine:
erzeugt Objekt (einer Ausnahmeklasse)

z. B. Objekt der Klasse NumberFormatException:

?

```
% java Ausnahme a1
java.lang.NumberFormatException:     a1
   at java.lang.Integer.parseInt
    at Ausnahme.makeIntFromString
     at Ausnahme.main
```


Wie kann man sich Objekte zur Beschreibung von Ausnahmen vorstellen?

Java-VM-Maschine:
erzeugt Objekt (einer Ausnahmeklasse)

z. B. Objekt der Klasse NumberFormatException:

Ausnahmeobjekt:
enthält alle Daten über eine Ausnahme, so dass diese Informationen ausgegeben werden können.

```
% java Ausnahme a1
java.lang.NumberFormatException:     a1
   at java.lang.Integer.parseInt
    at Ausnahme.makeIntFromString
     at Ausnahme.main
```


Wie kann man sich Objekte zur Beschreibung von Ausnahmen vorstellen?

Java-VM-Maschine:
erzeugt Objekt (einer Ausnahmeklasse)

Objekt der Klasse NumberFormatException:

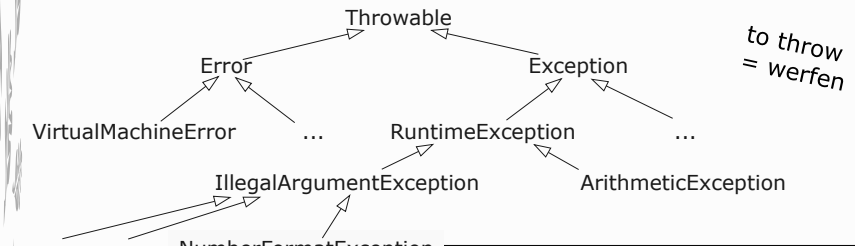
Methoden: NumberFormatException(String details)
getMessage()
printStrackTrace()
FillInStackTrace()

Daten:
Art: NumberFormatException
Details: a1
Stack-Trace: Integer.parseInt
makeIntFromString
main

```
% java Ausnahme a1
java.lang.NumberFormatException: a1
   at java.lang.Integer.parseInt
    at Ausnahme.makeIntFromString
     at Ausnahme.main
```

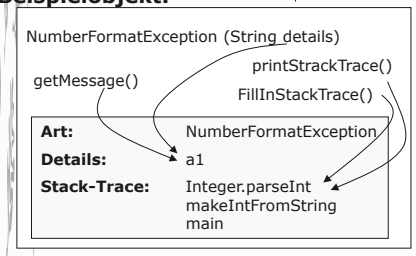

Ausnahme-Klassen: Hierarchie

(API: in java.lang)



to throw = werfen

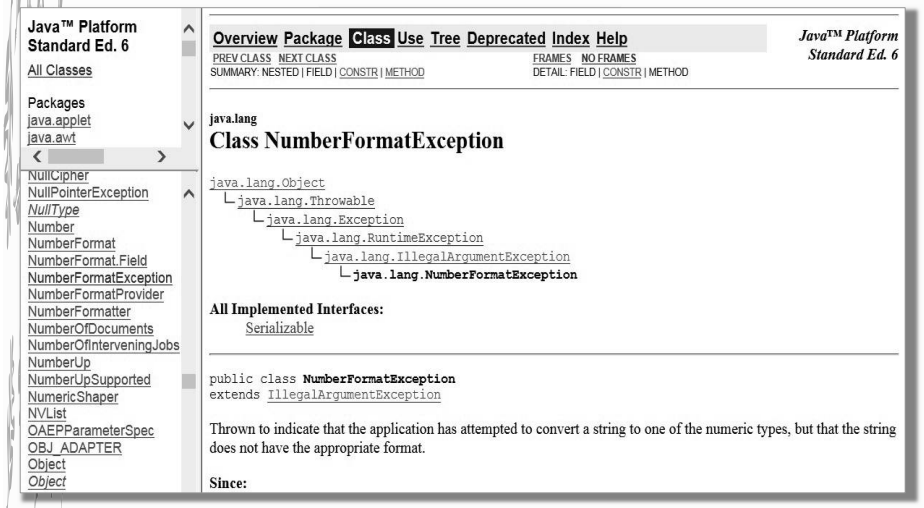
Beispielobjekt:



je Art einer Ausnahme: spezielle Klasse

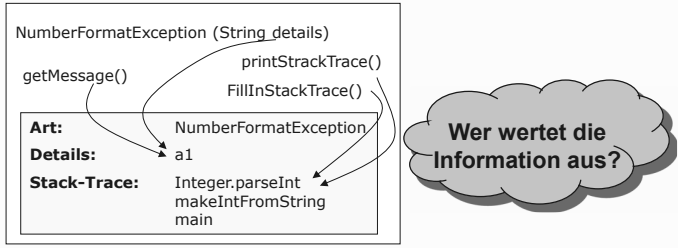
Throwable:	allgemeine Ausnahmeklasse
Error:	schwerer Fehler (i.allg. nicht behebbar)
Exception:	behandelbar

API-Klasse NumberFormatException



Ausnahmebehandlung bedeutet: Auswertung von erzeugten Ausnahmeobjekten

Objekt der Klasse NumberFormatException

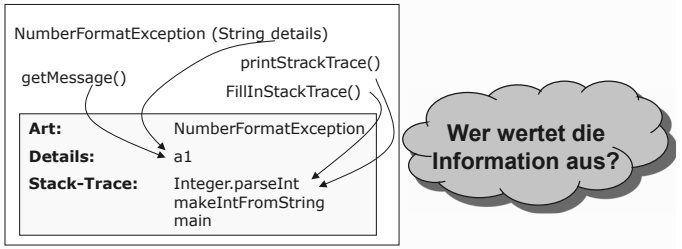


- **Java-Standard-Ausnahmebehandler (JVM):**
  - Ausgabe der im Ausnahmeobjekt gespeicherten Informationen (s.u.)
  - Abbruch des Programms

```
% java Ausnahme a1
java.lang.NumberFormatException:     a1
    at java.lang.Integer.parseInt
     at Ausnahme.makeIntFromString
      at Ausnahme.main
```


Ausnahmebehandlung bedeutet: Auswertung von erzeugten Ausnahmeobjekten

Objekt der Klasse NumberFormatException



- **Java-Standard-Ausnahmebehandler (JVM):**
  - Ausgabe der im Ausnahmeobjekt gespeicherten Informationen (s.o.)
  - Abbruch des Programms
- **Nutzerdefinierte Ausnahmebehandlung:**

try-catch - Anweisung

Nutzerdefinierte Ausnahmebehandlung in Java:

try-catch-Anweisung

try–catch: ein Beispiel

TryCatch.java

```
class TryCatch   {
   public static void main (String[] args)   {
      try   {
         int i = Integer.parseInt(args[0]);
         System.out.println("i = " + i);
      } catch (NumberFormatException e)   {
         System.out.println("Als Kommandozeilen-
           Argument wird ein int-Wert benoetigt");
      }
   }
}
```

try: Versuche eine normale Abarbeitung

catch: Falls normale Abarbeitung misslungen ist, „fange“ die Ausnahmesituation ab.

try–catch: ein Beispiel

TryCatch.java

```
class TryCatch   {
   public static void main (String[] args)   {
      try   {
         int i = Integer.parseInt(args[0]);
         System.out.println("i = " + i);
      } catch (NumberFormatException e)   {
         System.out.println("Als Kommandozeilen-
           Argument wird ein int-Wert benoetigt");
      }
   }
}
```

Kritische Anweisungen →Ausnahme-Situation möglich

Behandlung einer Ausnahmesituation → und danach: normale Weiterarbeit

try–catch: Wirkung im Detail

```
class TryCatch   {
   public static void main (String[] args)   {
      try   {
         int i = Integer.parseInt(args[0]);
         System.out.println("i = " + i);
      }
      catch (NumberFormatException e)   {
         System.out.println("Als Kommandozeilen-
                  Argument ...");
      }
   }
```

try-Block: Ausnahme möglich

→ Evtl. Ausnahmeobjekt o vom Typ t (Ausnahme-Klasse) erzeugt:

Falls t zum Typ des Parameters e von catch passt:
- Objekt o an Parameter e gebunden
- catch–Anweisung ausgeführt

Rolle von e?

→ Programmierer bestimmt, was nach einer Ausnahme (hier: NumberFormatException) passiert, **und das Programm wird regulär fortgesetzt !**

try–catch: Aufruf

```
class TryCatch   {
   public static void main (String[] args)   {
      try   {
         int i = Integer.parseInt(args[0]);
         System.out.println("i = " + i);
      } catch (NumberFormatException e)   {
         System.out.println("Als Kommandozeilen-
           Argument wird ein int-Wert benötigt");
      }
   }
```

```
% java TryCatch 3
i = 3
```

Ausgabe?

```
% java TryCatch a1
Als Kommandozeilen-Argument wird ein int-Wert benötigt
% java TryCatch
java.lang.ArrayIndexOutOfBoundsException: 0
 ...
```

falsches Argument

fehlendes Argument

Diese Ausnahme nicht behandelt

Behandlung mehrerer Ausnahmen

TryCatchAll.java

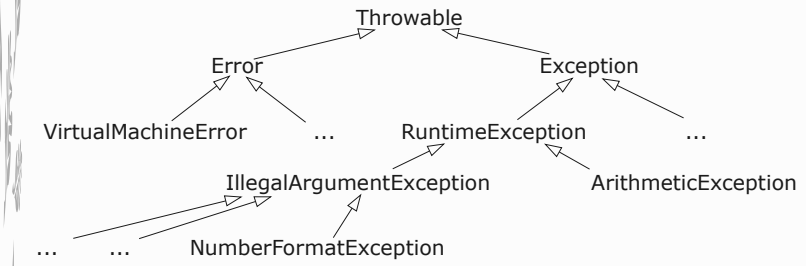
```
class TryCatchAll   {
   public static void main (String[] args)   {
       try   {
          int i = Integer.parseInt(args[0]);
          System.out.println("i = " + i);
       }
       catch (NumberFormatException e)   {
          System.out.println("Als Kommandozeilen ...");
       }
       catch (ArrayIndexOutOfBoundsException e)   {
          System.out.println("Aufruf mit einem Par.");
       }
       catch (Throwable e)   {
          e.printStackTrace();
       }
}
```

Alle Ausnahmen sind kompatibel zu Throwable

Methode auf Ausnahmeobjekt angewendet

Unterschied: letztes catch ←→ Interpreter?

Kompatibilität von Ausnahmen



Regel:
Ausnahme-Unterklassen sind immer kompatibel zur erwarteten Klasse, d.h. zu Throwable sind alle Ausnahmen kompatibel

je Art einer Ausnahme: spezielle Klasse	
Throwable:	allgemeinste Ausnahmeklasse
Error:	schwerer Fehler (i.allg. nicht behebbar)
Exception:	behandelbar

try-catch: reguläre Weiterarbeit

```
class TryCatchAll   {
   public static void main (String[] args)   {
       try   {  ...
       }
       catch (NumberFormatException e)   {...
       }
       catch (ArrayIndexOutOfBoundsException e)   {...
       }
       catch (Throwable e)   {  ...
       }
       System.out.println ("Programm ordentlich beendet");
   }
}
```

Mit und ohne Ausnahme: Ausschrift kommt

Ohne try-catch:
Programm nach Auftreten einer Ausnahme (irregulär) beendet

finally-Block

- Am Ende: finally-Code immer ausgeführt (mit / ohne Ausnahme)
- Idee: gemeinsamer Code für Abschlussaktivitäten (mit/ohne Ausnahmen)

Finally.java

```
class Finally   {
   public static void main (String[] args)   {
       int i = 1000, j = 0;
       try   {  i /= j;
       } catch (ArithmeticException e)   {
          System.out.println(e);
       } finally  {
          System.out.println (i + " / 0 undef.");
       }
   }
}
```

Hinweis kommt immer

```
% java Finally
. . .
1000 / 0 undef.
```

Verschachtelte try-Anweisungen

TryInTry.java

```
try   {
   try   {
      int x = Integer.parseInt(args[0]);
      }
   catch (NumberFormatException e)   {
      System.out.println("Innen: " + e);
   }
}
catch (Throwable e)   {
   System.out.println ("Aussen: " + e);
}
```

Argument kein 'int'

kein Argument

Falls Ausnahme im inneren catch nicht passt:

Suche umgebendes try-catch

(bzw. try-catch einer aufrufenden Methode)

Beispiel: sicheres Einlesen von Zahlen

KeyboardTry.java

```
public static int intEinlesen() {
  int zahl = 0; boolean ok = false;

  System.out.print("Zahl eingeben: ");
  while (!ok)   {
     try   {
        ok = true;
        zahl = Keyboard.readInt();
     }
     catch ( NumberFormatException e )   {
        System.out.print("Keine Zahl!! Noch einmal:");
        ok = false;
     }
  }
  return zahl;
}
```


Nutzerdefinierte Erzeugung von Ausnahmen in Java:

throw-Anweisung

Wer erzeugt Ausnahmeobjekte?

```
class Ausnahme   {
   public static void main (String[] args)   {
      int i = Integer.parseInt(args[0]);
      System.out.println("i = " + i);
   }
}
```

Hier: Auswertung von Ausnahme-objekten

Frage: Wer hat sie vorher erzeugt?

```
% java Ausnahme 3
i = 3

% java Ausnahme
java.lang.ArrayIndexOutOfBoundsException: 0
   at Ausnahme.main(Compiled Code)

% java Ausnahme a1
java.lang.NumberFormatException: a1
   at java.lang.Integer.parseInt
    at Ausnahme.main
```

Wer erzeugt Ausnahmen?

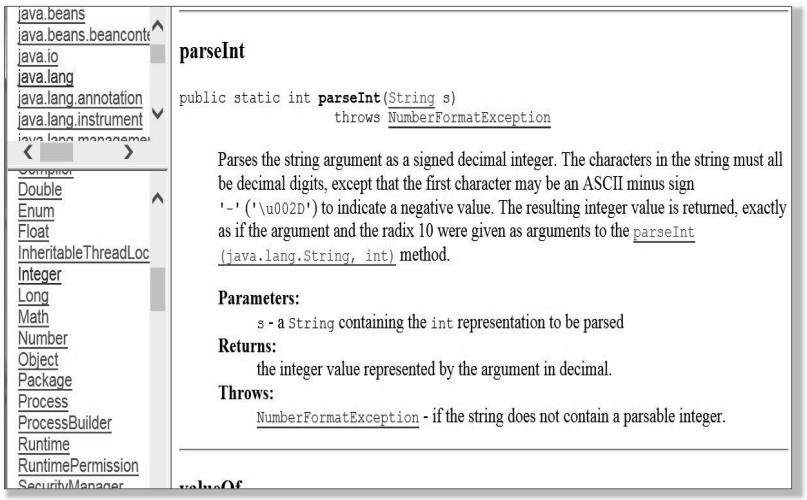
- Virtuelle Java-Maschine (Java-Interpreter, Laufzeitsystem):
  z.B. - Klasse nicht gefunden
  - falscher Array-Index
- Methoden des Java-API
  z. B. Integer.parseInt (in Java implementiert)

```
public static int parseInt (String s)
        throws NumberFormatException
```

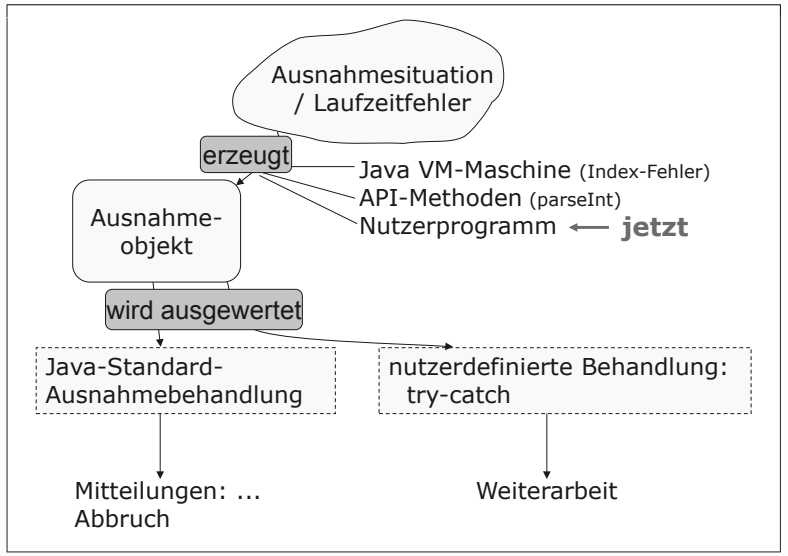
parseInt wertet s aus und erzeugt ggf. Ausnahme

- Nutzer-Programm:
  Throw-Anweisung

API: parseInt erzeugt Ausnahme



Ausnahmen: erzeugen + behandeln



Nutzerdefinierte Ausnahmeklasse mit throw-Anweisung

Oeffnungszeit.java

```
import java.util.*;

class Wochenende extends Exception {
        Wochenende(String text) {
            super(text);
        }
}

class Oeffnungszeit {

    static void heuteOffen () throws Wochenende {
        int tag = Calendar.getInstance().get(Calendar.DAY_OF_WEEK);
        if (tag == Calendar.SUNDAY)        // 1
            throw new Wochenende("Sonntag");
        if (tag == Calendar.SATURDAY)      // 7
            throw new Wochenende("Samstag");
        System.out.println("Heute geoeffnet.");
    }

    public static void main(String[] args) {
        try {
           heuteOffen();
        } catch (Wochenende ex) {
            System.out.println(ex.getMessage() + "s geschlossen.");
        }
    }
}
```

Nutzerdefinierte Ausnahmeklasse mit throw-Anweisung

```
import java.util.*;

class Wochenende extends Exception {
        Wochenende(String text) {
            super(text);
        }
}
```

```
class Oeffnungszeit {

    static void heuteOffen () throws Wochenende {
        int tag = Calendar.getInstance().get(Calendar.DAY_OF_WEEK);
        if (tag == Calendar.SUNDAY)         // 1
            throw new Wochenende("Sonntag");
        if (tag == Calendar.SATURDAY)       // 7
            throw new Wochenende("Samstag");
        System.out.println("Heute geoeffnet.");
    }

    public static void main(String[] args) {
        try {
           heuteOffen();
        } catch (Wochenende ex) {
            System.out.println(ex.getMessage() + "s geschlossen.");
        }
    }
}
```

Normalfall:
Arbeitstage (Mo - Fr)
Ausnahme:
Wochenende (Sa, So)

Nutzerdefinierte Ausnahmeklasse mit throw-Anweisung

```
import java.util.*;

class Wochenende extends Exception {
        Wochenende(String text) {
            super(text);
        }
}
```

Nutzerdefinierte Ausnahmeklasse

```
class Oeffnungszeit {

    static void heuteOffen () throws Wochenende {
        int tag = Calendar.getInstance().get(Calendar.DAY_OF_WEEK);
        if (tag == Calendar.SUNDAY)         // 1
            throw new Wochenende("Sonntag");
        if (tag == Calendar.SATURDAY)       // 7
            throw new Wochenende("Samstag");
        System.out.println("Heute geoeffnet.");
    }

    public static void main(String[] args) {
        try {
           heuteOffen();
        } catch (Wochenende ex) {
            System.out.println(ex.getMessage() + "s geschlossen.");
        }
    }
}
```

trow:
Erzeugung von Ausnahmen durch Anwenderprogramm

Auswertung des Ausnahmeobjekts: getmessage

Nutzerdefinierte Ausnahmeklasse mit throw-Anweisung

```
class Oeffnungszeit {

    static void heuteOffen () throws Wochenende {
        int tag = Calendar.getInstance().get(Calendar.DAY_OF_WEEK);
        if (tag == Calendar.SUNDAY)         // 1
            throw new Wochenende("Sonntag");
        if (tag == Calendar.SATURDAY)       // 7
            throw new Wochenende("Samstag");
        System.out.println("Heute geoeffnet.");
    }

    public static void main(String[] args) {
        try {
           heuteOffen();
        } catch (Wochenende ex) {
            System.out.println(ex.getMessage() + "s geschlossen.");
        }
    }
}
```

```
% java Oeffnungszeit
Heute geoeffnet
```

```
% java Oeffnungszeit
Samstags geschlossen
```

```
% java Oeffnungszeit
Sonntags geschlossen
```

Am Wochenende arbeitet das Programm anders als an Arbeitstagen

Nutzerdefinierte Ausnahmeklasse mit throw-Anweisung

```
class Oeffnungszeit {

    static void heuteOffen () throws Wochenende {
        int tag = Calendar.getInstance().get(Calendar.DAY_OF_WEEK);
```

```
C:\Bothe\GdP\Java_beispiele\TEIL_III>date
Aktuelles Datum: 11.01.2016
Geben Sie das neue Datum ein: (TT-MM-JJ)

C:\Bothe\GdP\Java_beispiele\TEIL_III>java Oeffnungszeit
Heute geoeffnet.
```

```
C:\Bothe\GdP\Java_beispiele\TEIL_III>date
Aktuelles Datum: 10.01.2016
Geben Sie das neue Datum ein: (TT-MM-JJ)

C:\Bothe\GdP\Java_beispiele\TEIL_III>java Oeffnungszeit
Sonntags geschlossen.
```

Am Wochenende arbeitet das Programm anders als an Arbeitstagen

Nutzerdefinierte Ausnahmen

```
class Wochenende extends Exception   {
   Wochenende(String text)   {
      super(text);   }
}
```

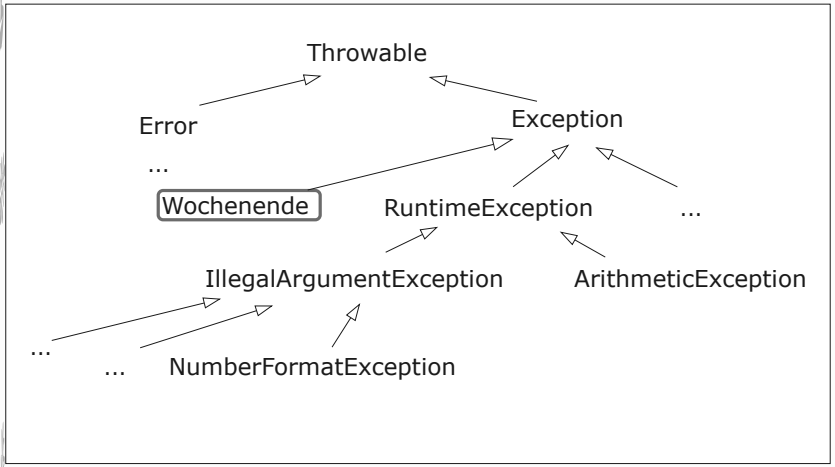
Oeffnungszeit.java

ex = new Wochende ("Sonntag")
ex.getMessage() → "Sonntag"

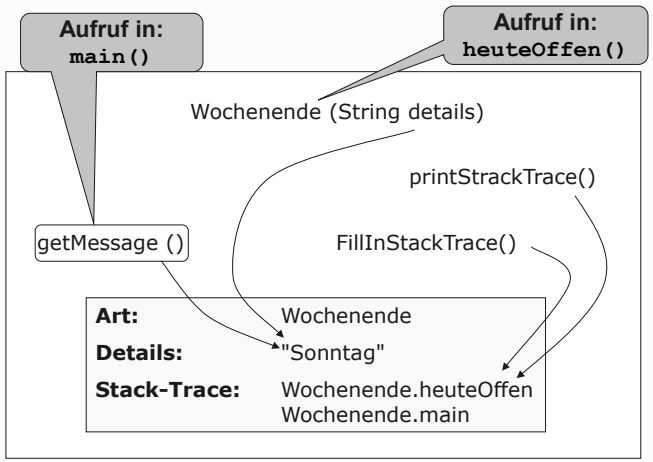
Java-API

```
class Exception   {

   public Exception (String s)
      // erzeugt E. mit Information s

   public String getMessage ()
      // gibt Information s zurueck
   ...
}
```


Neue Ausnahmeklasse: 'Wochenende'



Ausnahmeobjekt nach: throw new Wochenende ("Sonntag");



Anwendung: nutzerdefinierte Ausnahme

```
public static void main (String[] args)   {
   try   {
      heuteOffen();
   } catch (Wochenende ex)   {
      System.out.println ( ex.getMessage() +
                           "s geschlossen.");
   }
}
```

Normalfall

z. B. "Sonntag"

Ausnahmesituation

Methode erzeugt Ausnahme

```
static void heuteOffen() throws Wochenende {
   int tag = Calender ... // API-Klasse
   if (tag == Calendar.SUNDAY)                  // 1
      throw new Wochenende("Sonntag");
   if (tag == Calendar.SATURDAY)                // 7
      throw new Wochenende("Samstag");
   System.out.println("Heute geoeffnet.");
}
```

Im Ausnahmefall: nicht erreicht

API-Klasse Calender: liefert aktuellen Wochentag

```
static void heuteOffen () throws Wochenende {

   int tag = Calendar.getInstance().get(Calendar.DAY_OF_WEEK);

   if (tag == Calendar.SUNDAY)                  // 1
        throw new Wochenende("Sonntag");

   if (tag == Calendar.SATURDAY)                // 7
        throw new Wochenende("Samstag");

   System.out.println("Heute geoeffnet.");
}
```

aktueller Wochentag geliefert