Ausnahmen (Exceptions)

Herkömmliche Fehlerbehandlung

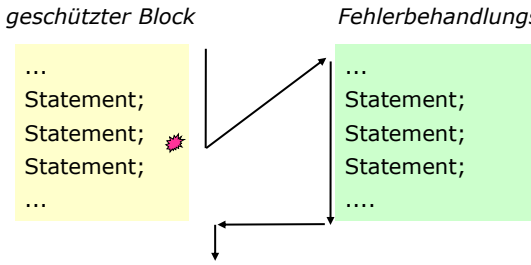
Jede Methode liefert einen Fehlercode

```
int result;
result = p(...);
if (result == ok) {
    result = q(...);
    if (result == ok) {
        result = r(...);
        if (result == ok) {
            ...
        } else error(...);
    } else error(...);
} else error(...);
```

Problem

- Aufwändig: vor lauter Fehlerbehandlung sieht man eigentliches Programm nicht mehr.
- Abfragen des Fehlercodes kann vergessen werden.
- Abfragen des Fehlercodes oft aus Bequemlichkeit nicht durchgeführt.

Idee der Ausnahmebehandlung in Java



Ausnahmebehandler
(exception handler)

Wenn im geschützten Block ein Fehler (eine Ausnahme) auftritt:

- Ausführung des geschützten Blocks wird abgebrochen.
- Fehlerbehandlungscode wird ausgeführt.
- Programm setzt nach dem geschützten Block fort.

Try- Anweisung in Java

Jede Methode liefert einen Fehlercode

```
try {
    p(...);
    q(...);
     r(...);
} catch (Exception1 e) {
    error(...);
} catch (Exception2 e) {
    error(...);
} catch (Exception3 e) {
   error(...);
}
```

geschützter Block

Ausnahmebehandler

Vorteile

- Fehlerfreier Fall und Fehlerfälle sind sauberer getrennt.
- Man kann nicht vergessen, einen Fehler zu behandeln.
  (Compiler prüft, ob es zu jeder möglichen Ausnahme einen Behandler gibt).

Try- Anweisung in Java

Laufzeitfehler

werden von der Java - VM ausgelöst

- Division durch 0 — *ArithmeticException*
- Zugriff über Null- Zeiger — *NullPointerException*
- Indexüberschreitung — *ArrayIndexOutOfBoundsException*

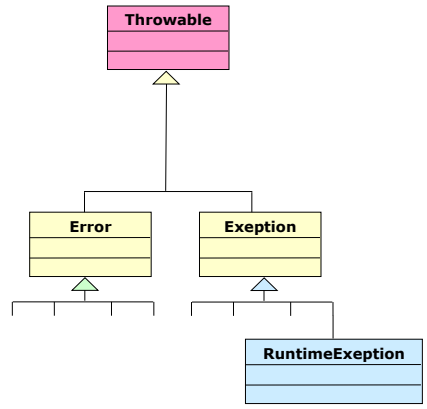
wenn sie nicht behandelt werden, stürzt das Programm mit einer Fehlermeldung ab

Geprüfte Ausnahmen

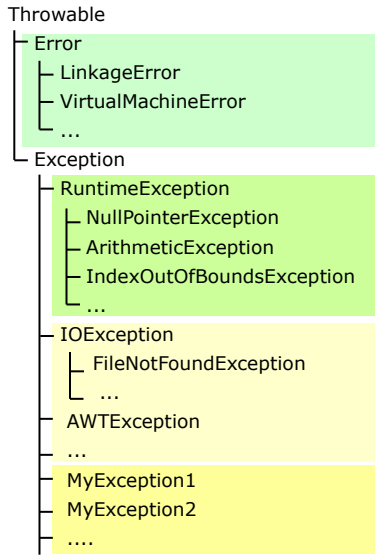
- vordefinierte Ausnahmen z. B. FileNotFoundException
- selbst definierte Ausnahmen z. B. MyException

Compiler prüft, ob sie abgefangen werden.

Hierarchie der Ausnahmeklassen



Hierarchie der Ausnahmeklassen



Fehler, die nicht am Programm liegen
(müssen / können nicht abgefangen werden)

von der VM ausgelöste Fehler
(müssen nicht abgefangen werden)

vordefinierte Ausnahmen
(müssen abgefangen werden)

selbst definierte Ausnahmen
(müssen abgefangen werden)

Ausnahmen als Objekte codiert

Fehlerinformationen stehen in einem Ausnahme-Objekt.

```
class Exception extends Throwable {
    Exception( String msg) {...}        // erzeugt neues Ausnahmeobjekt mit Fehlermeldung
    String getMessage() {...}           // liefert gespeicherte Fehlermeldung
    String toString() {...}             // liefert Art der Ausnahme und gespeicherte Fehlermeldung
    void printStackTrace() {...}        // gibt Methodenaufrufkette aus
    ...
}
```

Eigene Ausnameklasse (speichert Informationen über speziellen Fehler).

```
class MyException extends Exception {
   private int errorCode;
   MyException( String msg, int errorCode) {...}
   int getErrorCode() {...}
    // toString(), printStackTrace(), ... von Exception geerbt
}
```

Throw- Anweisung

Löst eine Ausnahme aus

```
throw new MyException(" invalid operation", 17);
```

```
if ( Ausnahme )  throw new  SomeException();
```

"Wirft" ein Ausnahmeobjekt mit entsprechenden Fehlerinformationen:

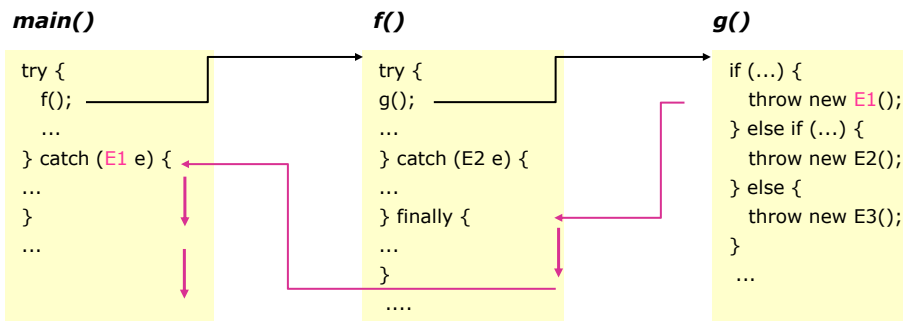
- bricht normale Programmausführung ab,
- sucht passenden Ausnahmebehandler (catch- Block),
- führt Ausnahmebehandler aus und übergibt ihm Ausnahmeobjekt als Parameter,
- setzt nach try- Anweisung fort, zu der der catch- Block gehört.

try, catch- Blöcke und finally- Block

```
try {
   ...
    if ( Ausnahme )  throw new  MyException();
   ...
} catch (MyException e) {
     Out. println( e. getMessage() + ", error code = ", + e. getErrorCode());
} catch (NullPointerException e) {
     ...
} catch (Exception e) {
     ...
} finally {
     ...
}
```

- Passender catch- Block wird an Hand des Ausnahme- Typs ausgewählt.
- catch- Blöcke werden sequentiell abgesucht.
- Achtung: speziellere Ausnahme- Typen müssen vor Allgemeineren stehen.
- Am Ende wird (optionaler) finally- Block ausgeführt.
- egal, ob im geschützten Block ein Fehler auftrat oder nicht.

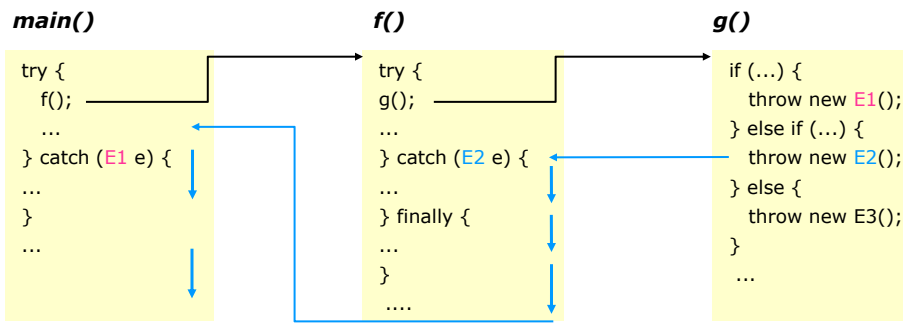
Ablauflogik bei Ausnahmen



throw new E1();

- keine try- Anweisung in *g()* bricht *g()* ab,
- kein passender catch- Block in *f()* führt finally- Block in *f()* aus und bricht *f()* dann ab,
- führt catch- Block für E1 in *main()* aus,
- setzt nach try- Anweisung in *main()* fort.

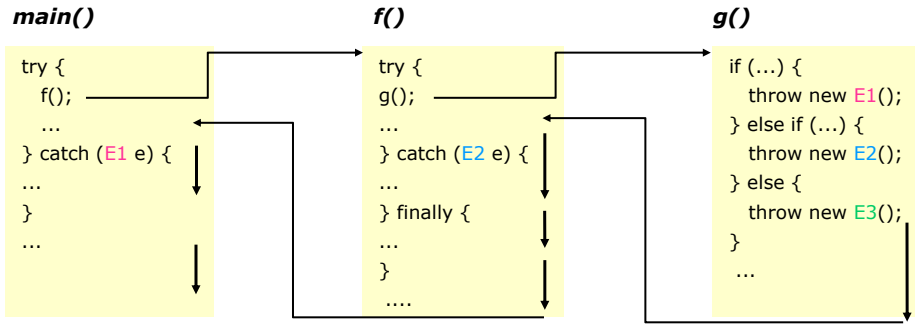
Ablauflogik bei Ausnahmen



throw new E2();

- keine try- Anweisung in *g()* bricht *g()* ab,
- führt catch- Block für *E2* in *f()* aus,
- führt finally- Block in *f()* aus,
- setzt nach try- Anweisung in *f()* fort.

Ablauflogik bei Ausnahmen



throw new E3();

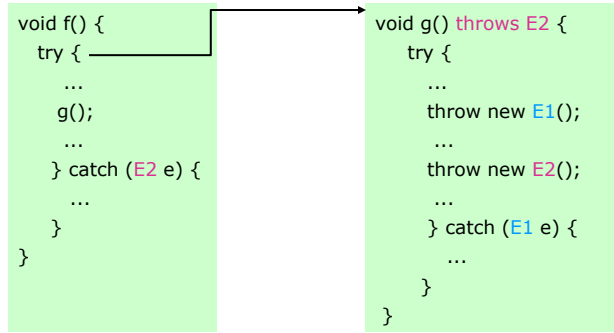
- Compiler meldet einen Fehler, weil *E3* nirgendwo in der Ruferkette abgefangen wird.

fehlerfreier Fall:

- *führt g()* zu Ende aus,
- führt try- Block in *f()* zu Ende aus,
- führt finally- Block in *f()* aus,
- setzt nach finally- Block in *f()* fort.

Spezifikation von Ausnahmen im Methodenkopf

Wenn eine Methode eine Ausnahme an den Rufer weiterleitet, muß sie das in ihrem Methodenkopf mit einer throw**s**- Klausel spezifizieren



Compiler weiß dadurch, daß *g()* eine *E2*- Ausnahme auslösen kann.

Wer *g()* aufruft, muß daher

- entweder *E2* abfangen,
- oder *E2* im eigenen Methodenkopf mit einer *throws- Klausel* spezifizieren.

Man kann nicht vergessen, eine Ausnahme zu behandeln!

Assert,
eine spezielle Ausnahme

assert: seit JAVA 1.4

Mit ***assert*** lassen sich Ausnahmen gezielt überprüfen.

```
assert  Expression : StringExpression;
```

löst bei false eine Ausnahme aus
und beendet das Programm
mit der StringExpression

Beispiel

```
static int ggt (int x, int y) {
    assert y!=0 :"Stopped: y == 0";
    int rest = x % y;
    while (rest != 0){
    x = y; y = rest;
    rest = x % y;
    }
    return y;
}
```

Die Berücksichtigung von assert und die Ausnahmen müssen beim Aufruf eines Programms explizit eingeschalten werden.

```
javac     MyClass.java
java -ea MyClass
```

in Paketen -ea:package
oder -da:package