In Software-Programmen können Fehler auftreten.

Ausnahmebehandlung

- Die erste Begegnung mit Java-Ausnahmen
- Was ist eine Ausnahme,
  und warum kümmere ich mich darum?
- Benutzen von Ausnahmen
- Vordefinierte Ausnahmeklassen
- Eigene Ausnahmeklassen
- Zusammenfassung

Die erste Begegnung mit Java-Ausnahmen

Ein einfaches Java-Programm:

```
class Apfel
{
  public static void main(String[] args)
  {
    int i = 1, j = 0, k;

    k = i / j;
  }
}
```

Übersetzen und Starten des Programms
führt zu folgendem Ergebnis:

```
> javac Apfel.java
> java Apfel

java.lang.ArithmeticException: / by zero
    at Apfel.main(Apfel.java:7)
```

Die erste Begegnung mit Java-Ausnahmen

Ein einfaches Java-Programm:

```
class Kiwi
{
  public static void main(String[] args)
  {
    int[] feld = new int[100];

    for (int i = 1; i <= feld.length; i++)
      feld[i] = i;
  }
}
```

Übersetzen und Starten des Programms
führt zu folgendem Ergebnis:

```
> javac Kiwi.java
> java Kiwi

java.lang.ArrayIndexOutOfBoundsException: 100
    at Kiwi.main(Kiwi.java:8)
```

Die erste Begegnung mit Java-Ausnahmen

Ein einfaches Java-Programm:

```
class Melone
{
  public static void main(String[] args)
  {
    String s = null;

    System.out.println(s.length());
  }
}
```

Übersetzen und Starten des Programms
führt zu folgendem Ergebnis:

```
> javac Melone.java
> java Melone

java.lang.NullPointerException
    at Melone.main(Melone.java:7)
```

Die erste Begegnung mit Java-Ausnahmen

Ein einfaches Java-Programm:

```
import java.io.*;

class EinlesenInt
{
  static int liesInt()
  {
    DataInput StdEingabe =
      new DataInputStream(System.in);

    return Integer.parseInt
      (StdEingabe.readLine());
  }
}
```

Übersetzen des Programms führt zu folgendem Ergebnis:

```
> javac EinlesenInt.java

EinlesenInt.java:11:
Exception java.io.IOException must be
caught, or it must be declared in the
throws clause of this method.
    return Integer.parseInt
      (StdEingabe.readLine());
                           ^
1 error
```

Was ist eine Ausnahme, und warum kümmere ich mich darum?

Der Begriff *Ausnahme* ist eine Abkürzung für "außergewöhnliches Ereignis".

> Definition:
> Eine *Ausnahme* ist ein Ereignis, das während der Ausführung eines Programms eintritt und den normalen Anweisungsablauf stört.

Wenn in einer Java-Methode eine Ausnahme auftritt, erzeugt die Methode ein *Ausnahmeobjekt* und übergibt dieses an das Java-Laufzeitsystem.

Das Laufzeitsystem ist dann verantwortlich für das Auffinden von Programmteilen, welche die Ausnahme behandeln.

Die Verwendung von Ausnahmeobjekten (kurz: Ausnahmen) zum Umgang mit Fehlern hat die folgenden Vorteile:

- Trennen von Fehlerbehandlung und "regulärem" Ablauf
- Fortpflanzen von Fehlern durch den Aufrufstapel
- Gruppieren und Differenzieren von Fehlertypen

Was ist eine Ausnahme, und warum kümmere ich mich darum?

Trennung von Fehlerbehandlung und "regulärem" Ablauf

Beispiel: ohne Fehlerbehandlung

```
readFile
{
  open the file;
  determine its size;
  allocate that much memory;
  read the file into memory;
  close the file;
}
```

Trennung von Fehlerbehandlung und "regulärem" Ablauf

Beispiel (Fortsetzung): mit "klassischer" Fehlerbehandlung

```
errorCodeType readFile
{
  initialize errorCode = 0;
  open the file;
  if (theFileIsOpen)
  {
    determine the length of the file;
    if (gotTheFileLength)
    {
      allocate that much memory;
      if (gotEnoughMemory)
      {
        read the file into memory;
        if (readFailed)
          errorCode = -1;
      }
      else
        errorCode = -2;
    }
    else
      errorCode = -3;
    close the file;
    if (theFileDidntClose && errorCode == 0)
      errorCode = -4;
    else
      errorCode = errorCode and -4;
  }
  else
    errorCode = -5;
  return errorCode;
}
```

Trennung von Fehlerbehandlung und "regulärem" Ablauf

Beispiel (Fortsetzung): mit "klassischer" Fehlerbehandlung

```
errorCodeType readFile
{
  initialize errorCode = 0;
  open the file;
  if (theFileIsOpen)
  {
    determine the length of the file;
    if (gotTheFileLength)
    {
      allocate that much memory;
      if (gotEnoughMemory)
      {
        read the file into memory;
        if (readFailed)
          errorCode = -1;
      }
      else
        errorCode = -2;
    }
    else
      errorCode = -3;
    close the file;
    if (theFileDidntClose && errorCode == 0)
      errorCode = -4;
    else
      errorCode = errorCode and -4;
  }
  else
    errorCode = -5;
  return errorCode;
}
```

Trennung von Fehlerbehandlung und "regulärem" Ablauf

Beispiel (Fortsetzung): mit Ausnahmebehandlung

```
readFile
{
  try
  {
    open the file;
    determine its size;
    allocate that much memory;
    read the file into memory;
    close the file;
  }
  catch (fileOpenFailed)
  {
    doSomething;
  }
  catch (sizeDeterminationFailed)
  {
    doSomething;
  }
  catch (memoryAllocationFailed)
  {
    doSomething;
  }
  catch (readFailed)
  {
    doSomething;
  }
  catch (fileCloseFailed)
  {
    doSomething;
  }
}
```

Was ist eine Ausnahme, und warum kümmere ich mich darum?

Fortpflanzen von Fehlern durch den Aufrufstapel

Beispiel: ohne Fehlerbehandlung

```
method1
{
  call method2;
}

method2
{
  call method3;
}

method3
{
  call readFile;
}
```

Fortpflanzen von Fehlern durch den Aufrufstapel

Beispiel (Fortsetzung): mit "klassischer" Fehlerbehandlung

```
method1
{
  errorCodeType error;
  error = call method2;
  if (error)
    doErrorProcessing;
  else
    proceed;
}

errorCodeType method2
{
  errorCodeType error;
  error = call method3;
  if (error)
    return error;
  else
    proceed;
}

errorCodeType method3
{
  errorCodeType error;
  error = call readFile;
  if (error)
    return error;
  else
    proceed;
}
```

Fortpflanzen von Fehlern durch den Aufrufstapel

Beispiel (Fortsetzung): mit Ausnahmebehandlung

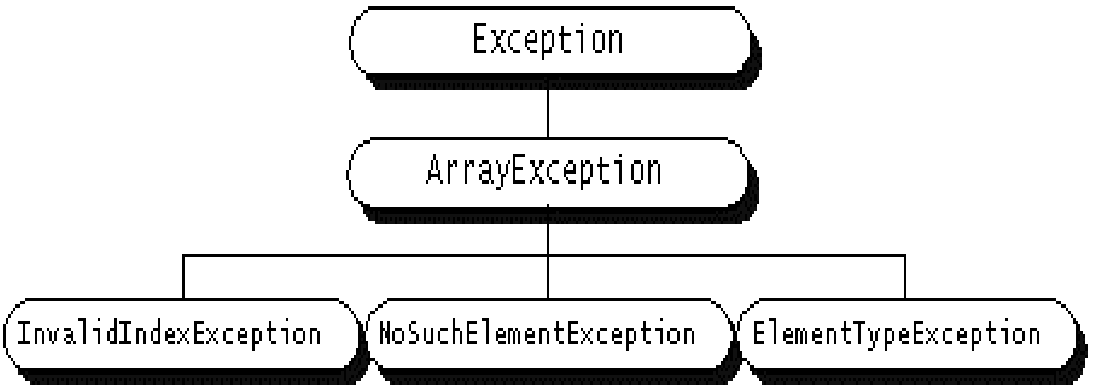
```
method1
{
  try
  {
    call method2;
  }
  catch (exception)
  {
    doErrorProcessing;
  }
}

method2 throws exception
{
  call method3;
}

method3 throws exception
{
  call readFile;
}
```

Was ist eine Ausnahme, und warum kümmere ich mich darum?

Gruppieren und Differenzieren von Fehlertypen

Beispiel:

Eine Gruppe von Ausnahmen, von denen jede einen spezifischen Fehlertyp darstellt, der bei Feldveränderung eintreten kann.



Abfangen von Ausnahmen wegen ungültigem Index:

```
catch (InvalidIndexException iie) { ... }
```

Abfangen *aller* Feldausnahmen:

```
catch (ArrayException ae) { ... }
```

Benutzen von Ausnahmen

Ablauf der Ausnahmebehandlung:

1. Ein Block von Anweisungen versucht (englisch ***try***) die Erledigung einer Aufgabe.

2. Wenn ein Fehler festgestellt wird, der nicht behoben werden kann, wirft (englisch ***throw***) der Block eine Ausnahme (englisch *exception*) aus.

3. Die Ausnahme wird von einem Ausnahmebehandler aufgefangen (englisch ***catch***), der den Fehler bearbeitet.

4. Es kann ein so genannter ***finally***-Block angegeben werden, der am Ende jedes *try*-Blocks auf jeden Fall ausgeführt wird.

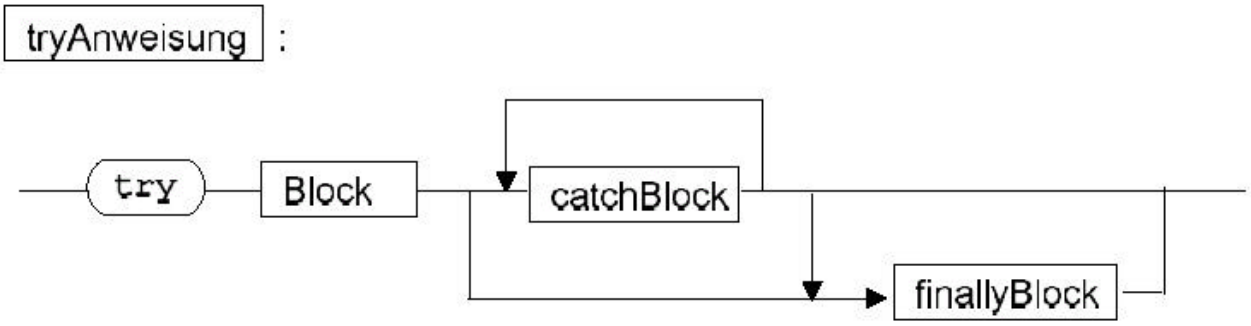
Benutzen von Ausnahmen

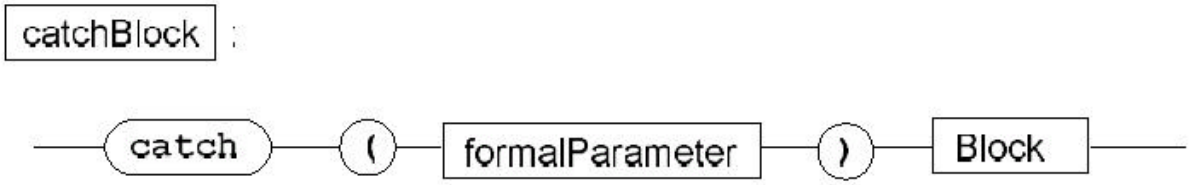
syntaktische Struktur der Ausnahmebehandlung:

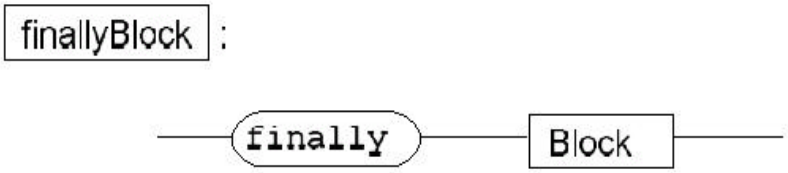
```
try
{
  // Normalerweise läuft dieser Code ohne Probleme
  // vom Anfang des Blocks bis zum Ende durch.
  // Manchmal können aber auch Ausnahmen ausgelöst
  // werden.
  ...
  if (...)
    throw new SomeException(...);
  ...
  if (...)
    throw new AnotherException(...);
  ...
}

catch (SomeException e1)
{
  // Behandle ein Ausnahme-Objekt e1 vom Typ
  // SomeException oder einer Unterklasse dieses Typs.
}

catch (AnotherException e2)
{
  ...
}

finally
{
  // Dieser Code wird am Ende jedes
  // try-Blocks auf jeden Fall ausgeführt.
}
```

Benutzen von Ausnahmen

Syntaxdiagramme







Benutzen von Ausnahmen

Auslösen von Ausnahmen

implizit:

- das Programm führt etwas Unzulässiges aus

  Beispiele:

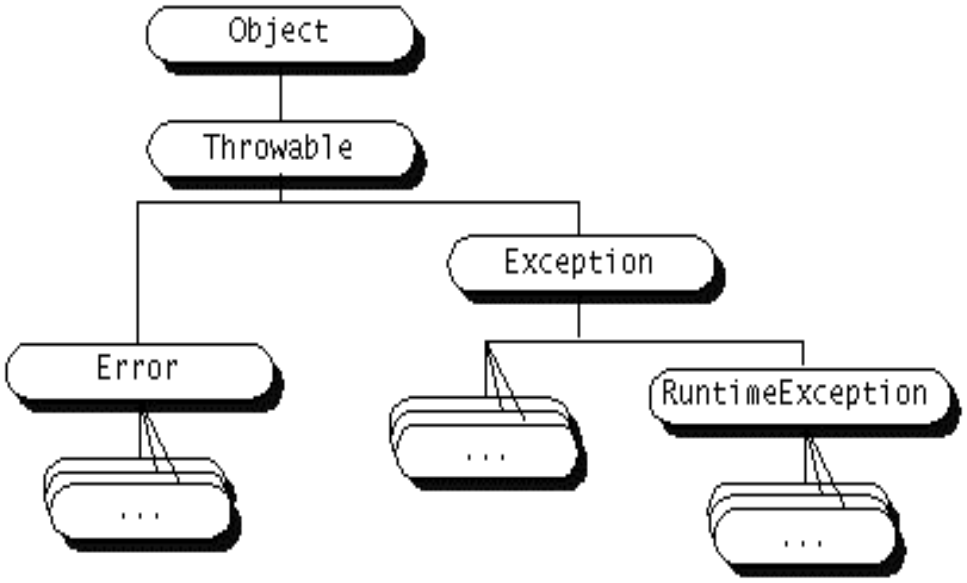
  -
  -
  -

explizit:

- durch die `throw`-Anweisung:

  `throw` *`Ausnahmeobjekt`*`;`

Ausnahmeobjekte sind Exemplare einer Klasse, die von `java.lang.Throwable` abgeleitet ist.



Benutzen von Ausnahmen

Auslösen von Ausnahmen

Der Ausnahme-Konstruktor kann ein `String`-Argument haben, das mit der Methode `getMessage()` erfragt werden kann.

Beispiel:

```
try
{
  throw new Exception1("Verflixt!")
}

catch (Exception1 e1)
{
  System.out.println(e1.getMessage());
}
```

Benutzen von Ausnahmen

Abfangen und Behandeln von Ausnahmen

Ausnahmebehandler (`catch`-Blöcke) werden durch *Typen* unterschieden.

Es wird der *erste* Ausnahmebehandler ausgeführt,

- dessen Typ genau mit dem Typ der signalisierten Ausnahme (`throw`-Anweisung) übereinstimmt
- oder dessen Typ eine Basisklasse der signalisierten Ausnahme ist.

Beispiel:

```
try {
  someReallyExceptionalMethod();
} catch (NullPointerException n) {
  ...
} catch (RuntimeException r) {
  ...
} catch (IOException i) {
  ...
} catch (MyFirstException m) {
  ...
} catch (Exception e) {
  ...
} catch (Throwable t) {
  ...
}
```

Benutzen von Ausnahmen

Abfangen und Behandeln von Ausnahmen

geschachtelte Ausnahmebehandlung:

- Falls für eine ausgelöste Ausnahme in diesem Anweisungsblock kein Ausnahmebehandler vorhanden ist, so wird die Ausnahme in den umfassenden Block weitergegeben
- Falls in dieser Methode kein umfassender Block mehr vorhanden ist ("*Die Methode duckt sich.*"), so wird im Block der Aufrufstelle nach einem Ausnahmebehandler gesucht.
- usw.

Falls kein Ausnahmebehandler gefunden wird, wird das Java-Programm mit Fehlermeldung und Stacktrace abgebrochen.

Benutzen von Ausnahmen

Abfangen und Behandeln von Ausnahmen

Beispiel: geschachtelte Ausnahmebehandlung

```
class X
{
   void xx() throws Exception3
   {
      try
      {
         try { throw new ...; }
         catch (Exception1 e1)
         {
            ...
         }
      }
      catch (Exception2 e2)
      {
         ...
      }
   }
}

class Y
{
   void yy()
   {
      xobj = new X();
      try
      {
         xobj.xx();
      }
      catch (Exception3 e3)
      {
         ...
      }
   }
}
```

Benutzen von Ausnahmen

Abfangen und Behandeln von Ausnahmen

Ausnahmen

- entweder **abfangen**
- oder **weiterleiten**

Methoden müssen durch den `throws`-Zusatz eine Liste der Ausnahmen angeben, die sie nach "außen" auslösen können.

- Allerdings nur für so genannte "*geprüfte* Ausnahmen" (das sind solche, die nicht von `RuntimeException` oder von `Error` abgeleitet worden sind).

- Es können Ausnahmeklassen oder Oberklassen der tatsächlich ausgelösten Ausnahmen angegeben werden.

Beispiele:

```
public void open_file() throws IOException
{  ...  }

public void myfunc(int arg)
  throws MyException1, MyException2
{  ...  }
```

Benutzen von Ausnahmen

finally

Beispiel: Die Datei `f` muss auf jeden Fall geschlossen werden, gleichgültig, ob in `someReallyExceptionalMethod` ein Fehler auftritt oder nicht.

ohne `finally` :

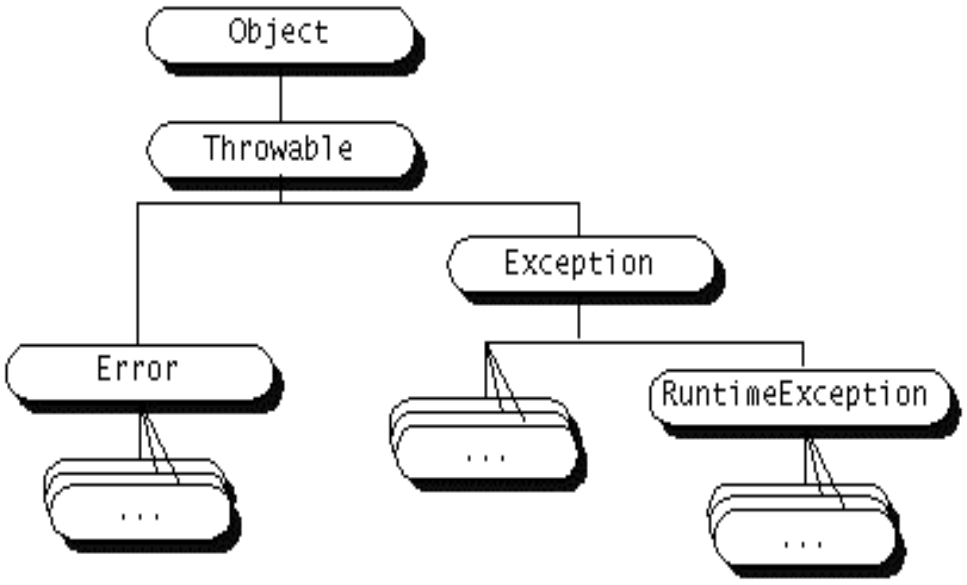
```
SomeFileClass f = new SomeFileClass();

if (f.open("a/file/name/path")) {
  try {
    someReallyExceptionalMethod();
    f.close();
  } catch (IOException e) {
    // deal with errors
    f.close();
  }
}
```

mit `finally` :

```
SomeFileClass f = new SomeFileClass();

if (f.open("a/file/name/path")) {
  try {
    someReallyExceptionalMethod();
  } catch (IOException e) {
    // deal with errors
  } finally {
    f.close();
  }
}
```

Vordefinierte Ausnahmeklassen

Ausnahmeobjekte sind Exemplare einer Klasse, die von `java.lang.Throwable` abgeleitet ist.



Vordefinierte Ausnahmeklassen

Die Klasse `Throwable` stellt die Basismethoden für Ausnahmen bereit.

```
public Throwable()
public Throwable(String message)
```

- Erzeugen ein `Throwable`-Objekt ohne bzw. mit Fehlermeldung.

```
public String getMessage()
```

- Liefert die beim Konstruktoraufruf angegebene Fehlermeldung bzw. eine leere Zeichenkette, wenn der parameterlose Konstruktor verwendet wurde.

```
public String toString()
```

- Liefert eine `String`-Darstellung des aktuellen Objekts.

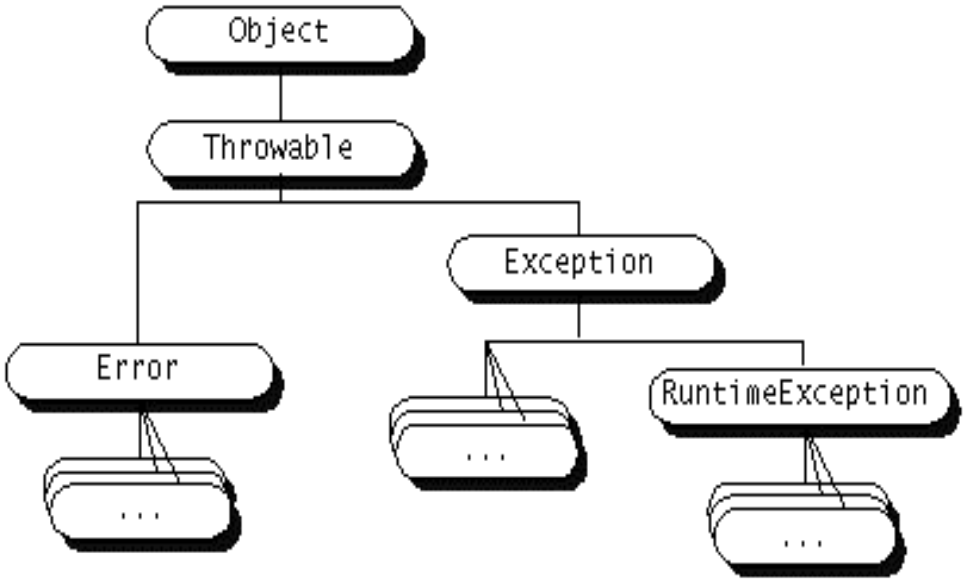
```
public void printStackTrace()
public void printStackTrace(PrintWriter s)
```

- Geben das aktuelle Objekt und den Stacktrace auf dem Bildschirm bzw. in den Ausgabestrom `s` aus.

```
public native Throwable fillInStackTrace()
```

- Trägt in das `Throwable`-Objekt die aktuelle Stackinformation ein.

Vordefinierte Ausnahmeklassen




```
public
class Exception extends Throwable {
  public Exception() { super(); }
  public Exception(String s) { super(s); }
}

public
class RuntimeException extends Exception {
  public RuntimeException() { super(); }
  public RuntimeException(String s) {super(s);}
}
```

Vordefinierte Ausnahmeklassen

sind z. B.:

`Error` (Bindefehler oder Fehler in der virtuellen Maschine)

- `LinkageError`
  - `IncompatibleClassChangeError`
  - `NoClassDefFoundError`
- `VirtualMachineError`
  - `InternalError`
  - `OutOfMemoryError`
  - `StackOverflowError`

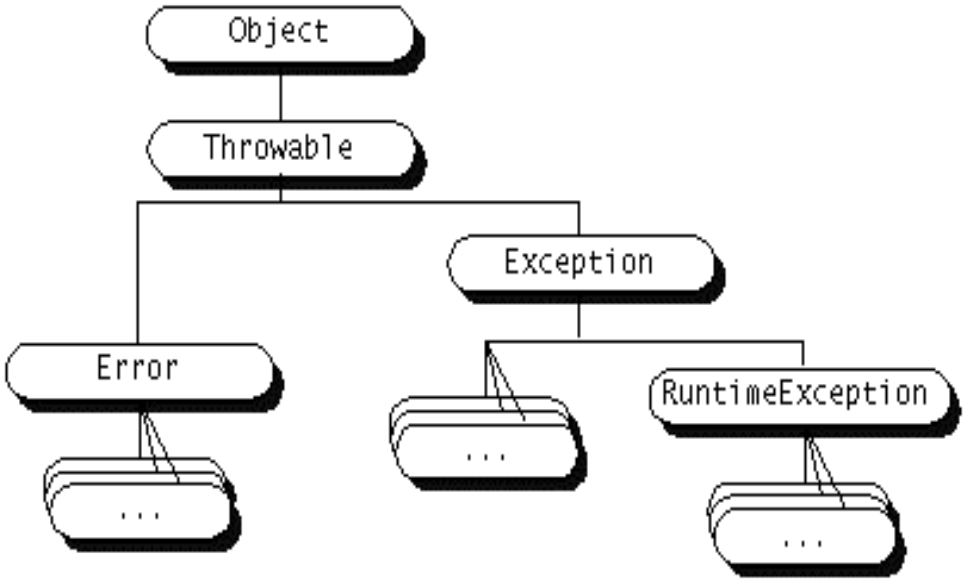
`RuntimeException` (meist Programmierfehler)

- `ArithmethicException`
- `ClassCastException`
- `IndexOutOfBoundsException`
- `NegativeArraySizeException`
- `NullPointerException`
- `NumberFormatException`

`IOException` (Fehler bei der Ein-/Ausgabe)

- `EOFException`
- `FileNotFoundException`
- `InterruptedIOException`
- `MalformedURLException`
- `ProtocolException`
- `UnknownHostException`
- `UnknownServiceException`

Eigene Ausnahmeklassen



Eigene Ausnahmeklassen sollten *nicht* von

- `Throwable`,
- `Error` oder
- `RuntimeException`

abgeleitet werden, sondern von

-
-

Beispiel:

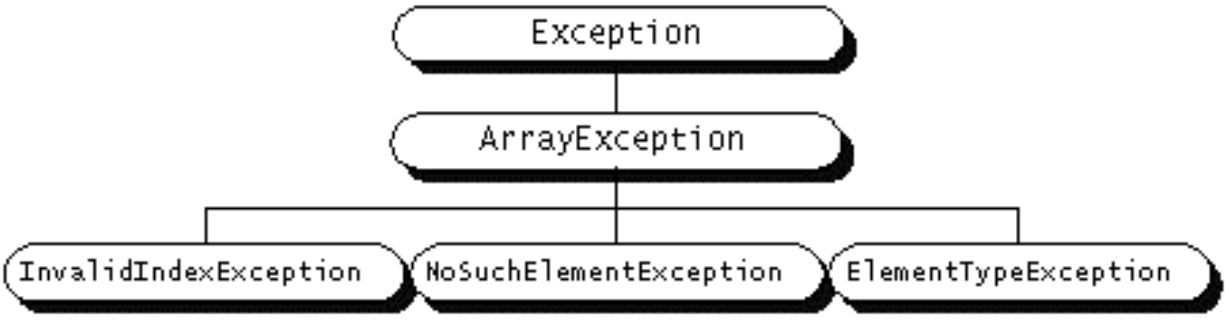
```
public class MyException extends Exception
{




}
```

Eigene Ausnahmeklassen

Gruppieren und Differenzieren von Fehlertypen
durch hierarchische Anordnung von Ausnahmeklassen

Beispiel:

Fehler, die beim Zugriff auf ein Feld auftreten können



Abfangen von Ausnahmen wegen ungültigem Index:

```
catch (InvalidIndexException iie) { ... }
```

Abfangen *aller* Feldausnahmen:

```
catch (ArrayException ae) { ... }
```

Zusammenfassung

Eine Ausnahme ist ein Signal, das ein außergewöhnliches Ereignis anzeigt.

- Durch `throw` wird ein solches Ereignis signalisiert:

```
throw new MyException
  ("my exceptional condition occurred");
```

- Durch `catch` wird eine Ausnahme behandelt: `try`/`catch`/`finally`-Anweisung

```
try { ... }
catch (SomeException s) { ... }
catch (AnotherException a) { ... }
finally { ... }
```

- Methoden müssen eventuelle Ausnahmen behandeln oder deren Typ spezifizieren:

```
public void myfunc(int arg)
  throws MyException1, MyException2 { ... }
```

- Die Definition eigener Ausnahmeklassen ist möglich:

```
class MyException extends Exception {
  public MyException() { super(); }
  public MyException(String s) { super(s); }
}
```

Zusammenfassung

Die Ausnahmebehandlung in Java erlaubt eine klare Trennung der Programmteile für den Normalfall und den Ausnahmefall.

Die Ausnahmebehandlung ermöglicht die

- **Erkennung** und die
- **Behandlung**

von Fehlern.

Die Fehlererkennung ist in der Regel einfach, die Behandlung schwierig und häufig genug unmöglich.

Die Art der Fehlerbehandlung hängt auch davon ab, wie sicherheitskritisch der Einsatz der Software ist.

Beispiel: Die Fehlermeldung

```
Index-Fehler im Array controlParameters,
Index = 2198, Max = 84
```

- in einem Textverarbeitungsprogramm
- in der Software eines in der Luft befindlichen Flugzeugs