Bezeichner

- Groß- und Kleinschreibung wird unterschieden
- erlaubt Buchstaben, Ziffern, weitere Zeichen aus dem Unicode-Alphabet (Unterstrich, Währungszeichen, griechische Buchstaben)
- Am Anfang keine Ziffern

Variable
variable
objekt1
~~**1objekt**~~
summe$
$summe
_variable

Namenskonventionen

Klasse, Interface: beginnt mit Großbuchstabe
sonst : beginnt mit Kleinbuchstabe
Paketnamen: nur Kleinbuchstaben

String
Runnable
objektX
java.util.Scanner;

Schlüssel-wörter

Datenbezug	Ablaufsteuerung	Objektbezug
boolean	assert	abstract
byte	break	class
char	case	enum
double	catch	extends
false	continue	final
final	default	implements
float	do	import
int	else	instanceof
long	finally	interface
short	for	native
static	(goto)	new
transient	if	null
true	return	package
void	switch	private
	synchronized	protected
	throw	public
	throws	strictfp
	try	super
	while	this
		volatile

elementare nummerische Datentypen und Boolean

Typ	Vorzeichen	Größe	Wertebereich
byte	ja	8 bit	-2^7 bis $2^7 - 1$ (-128...127)
short	ja	16 bit	-2^{15} bis $2^{15} - 1$ (-32768...32767)
int	ja	32 bit	-2^{31} bis $2^{31} - 1$ (-2147483648...2147483647)
long	ja	64 bit	-2^{63} bis $2^{63} - 1$ (-9223372036854775808...9223372036854775807)
float	ja	32 bit V: 1 bit E: 8 bit M: 23 bit	$-3.40282347 * 10^{38}$ bis $3.40282347 * 10^{38}$
double	ja	64 bit V: 1 bit E: 11 bit M: 52 bit	$-1.79769313486231570 * 10^{308}$ bis $1.79769313486231570 * 10^{308}$
boolean	-	8 bit	true/false

Quelle: https://javabeginners.de/Grundlagen/Datentypen/Primitive_Datentypen.php

elementare Datentypen

Definition:
`[Modifizierer] Typ name [= Initialisierung];`
`Typ n1, … ,nk;`

Es wird Datenelement angelegt, nicht nur Referenz

```
static int a=123;
long b=123L;
```

Referenztypen

Klassen
Interfaces
Array

```
public class X{}

X x=new X();
```

```
public class ElementareDaten {
     static byte b0;//wird auf 0 initialisiert

     public static void main(String[] args) {

          byte   b = 0177;
          short  s = 0x7fff;
          float  f = Float.MAX_VALUE;
          double d = Double.MAX_VALUE;
          char   c = 'j';
          boolean bu = true;


          System.out.println ("byte\tb0 = " + b0);
          System.out.println ("byte\tb = " + b);
          System.out.println ("short\ts = " + s);
          System.out.println ("char\tc = " + c);
          System.out.println ("float\tf = " + f);
          System.out.println ("double\td = " + d);
          System.out.println ("boolean\tbu = " + bu);
     }
}
```

```
byte      b0 = 0
byte      b = 127
short     s = 32767
char      c = j
float     f = 3.4028235E38
double    d = 1.7976931348623157E308
boolean   bu = true
```

Quelle: Prof. B. Steinbach, Vorlesung „Verteilte Software“

Array

Oracle Java Documentation: An *array* is a container object that holds a fixed number of values of a single type

- ist eine Referenz
- Das Objekt wird angelegt zur Laufzeit
- Elemente werden über Indizes angesprochen (0..Anzahl-1)

```
char[] wort={'A','b','r','a'};//char[] wort=new char[4];
for (int i = 0; i < wort.length; i++)
        System.out.print(wort[i] + " ");

Animal[]  animals=new Dog[5];
for (int i = 0; i < animals.length; i++)
        animals[i]=new Dog("Rex"+String.valueOf(i), "Dogge");
```

Wenn Array ein Objekt ist, was ist mit
wort.toString()?
animals.toString()?

vom Object geerbte liefert: className + @ + hex hashCode

```
System.out.println(Arrays.toString(animals));
```

```
public class ArrayBeispiel {
  final static int MAX = 4;

  public static void main(String args[]){
    int  i,
         x [],
         y [] = new int[MAX];

    x = new int[MAX];

    for (i = 0; i < MAX; i++) { x[i] = i; y[i] = 1;}

    aus(x); aus(y);
   }

 static void aus (int[] z){
     for (int i = 0; i < z.length; i++) System.out.print (z[i] + " ");
     System.out.println ();
     }
}
```

```
0 1 2 3
1 1 1 1
```

Array wird an die Methode als Referenz übergeben, Elemente werden nicht kopiert

```
public class MatrixBeispiel {
  final static int DIM1 = 4, DIM2 = 5;

  public static void main(String args[]){
     int i, j, matrix[][];

     matrix = new int [DIM1][DIM2];

    for (i = 0; i < matrix.length; i++)
        for (j = 0; j < matrix[i].length; j++)
           matrix[i][j] = i*10+j;

    System.out.println ("MATRIX");
    for (i = 0; i < DIM1; i++)
       { for (j = 0; j < DIM2; j++)
            System.out.print (matrix[i][j] + " ");
         System.out.println ();
       }
    }
}
```

```
MATRIX
0 1 2 3 4
10 11 12 13 14
20 21 22 23 24
30 31 32 33 34
```

```
int matrix[][];
int [] matrix [];
int [][] matrix;
```

String und „Verwandschaft“

Java nutzt Unicode!
jedes Zeichen der Welt eindeutig codierbar (120 000 Zeichen)
JVM: intern UTF-16
Zum Austausch der Texte wird üblicherweise UTF-8 verwendet

Beispiele	A	ß	東	𐐀
Unicode (hex)	U+0041	U+00DF	U+6771	U+10400
UTF-32 (4 Byte)	00000041	000000DF	00006771	00010400
UTF-16 (2 oder 4)	00 41	00 DF	67 71	D8 01 DC 00
UTF-8 (1, 2, 3 oder 4)	41	C3 9F	E6 9D B1	F0 90 90 80

Quelle: http://openbook.rheinwerk-verlag.de/javainsel/04_001.html

char ist ein Zeichen, aber es gibt noch Character? Was ist das und wozu?

- Wrapper – **Klasse** (sowie Integer, Double etc.)
- Viele Methoden lassen als Parameter nur von **Object** abgeleitete Klassen zu
- Außerdem verfügt über zahlreiche Konstanten und Konvertierungsfunktionen

```
java.lang.Character

Methoden:
static boolean isDigit (char ch)
static boolean isLetter (char ch)
static boolean isLetterOrDigit (char ch)
static boolean isLowerCase (char ch)
boolean isUpperCase (char ch)
static boolean isWhiteSpace (char ch)
//Leerzeichen, Zeilenvorschub, Return oder Tabulator
static char toUpperCase (char ch)
static char toLowerCase (char ch)
```

Bemerkung am Rande:
Double dd=53.23E6;
long l=dd.longValue();

Was ist String?

- Sammlung von Zeichen (char)
- unveränderbar, threadsicher (~~s[0]='A'~~)
- eine Symbiose zwischen dem String als Objekt und dem String als Datentyp
  (s. +-Operator, eine Ausnahme)

```
String s1 = "Sehr originell!";
String s2 = new String("Noch origineller!");
System.out.println (s1+" "+s2);

char[] charArray= {'\u0041','\u6771'};
String s=new String(charArray);
```

- ca. 30 Methoden

```
charAt(index:int):char
getBytes():byte[]
equals(object:Object):boolean
isEmpty():boolean
length():int
split(regex:String):String[]
valueOf(i:int):String
```

```
"".length()    // 0
"".isEmpty()   // true
" ".length()   // 1
" ".isEmpty()  // false
```

StringBuffer und StringBuilder sind veränderbar

```
public class Zeichenketten {
  static String static_text = "kühl ";

  public static void main(String args [])
  {
    String s1 = "kühler ";
    String s2;

    s2 = "am kühlsten ";
    System.out.println (static_text + s1 + s2);
    System.out.println (static_text.length() + s1.length() + s2.length()
     + " Zeichen insgesamt" );
  }
}
```

```
kuhl kühler am kühlsten
24 Zeichen insgesamt
```

weiteres zu String, StringBuffer und StringBuilder s. z. B.
Java ist auch eine Insel
http://openbook.rheinwerk-verlag.de

Operatoren

- Arithmetische Operationen
- Vergleichs und logische Operationen
- Inkrement und Dekrement
- Bedingungsoperator

  String s = x.isSelected() ? “ja“ : “nein“

- Bitverarbeitung
- Zuweisungen

  Mehrfachzuweisung: a=b=c;

  a+=10 entspricht a=a+10

Reihenfolge der Ausführung:

- Priorität: 15 Stufen
- innerhalb der Stufen - in der 2. Spalte

1	`() [] . lv++ lv--`	links nach rechts
2	`! ~ -unär +unär ++lv --lv`	rechts nach links
3	**`new`** `(type)`	
4	`* / %`	links nach rechts
5	`+ -`	links nach rechts
6	`<< >> >>>`	links nach rechts
7	`< <= > >=` **`instanceof`**	links nach rechts
8	`== !=`	links nach rechts
9	`&`	links nach rechts
10	`^`	links nach rechts
11	`|`	links nach rechts
12	`&&`	links nach rechts
13	`||`	links nach rechts
14	`?:`	rechts nach links
15	`= += -= *= /= %= ^= &= |= <<= >>= >>>=`	rechts nach links

Alternativen

```
if ( Bedingung ) Anweisung
if ( Bedingung ) Anweisung_1 else Anweisung_2
switch (Ausdruck)
  {
    case const_ausdruck_1 : Anweisungen
    case const_ausdruck_2 : Anweisungen
    :
    case const_ausdruck_n : Anweisungen
    default : Anweisungen
  }
```

Ergebnis des Ausdrucks:
byte, char, short, int

Iteration

```
while ( Bedingung ) Anweisung
do Anweisung while (Bedingung);
for ( Initialisierung; Bedingung; Ausdruck ) Anweisung
for ( Parameter: Container) Anweisung
```

```
short i1 = 33;
int   i2 = 58;
char  antwort;

if (i1 == i2)
  System.out.println (i1 + " gleich " + i2);
else
  System.out.println (i1 + " ungleich " + i2);


Scanner sc=new Scanner(System.in);
antwort=sc.next().charAt(0);

switch (antwort)
{
  case 'A': System.out.println ("wie Alfa"); break;
  case 'B': System.out.println ("wie Bravo"); break;
  case 'C': System.out.println ("wie Charlie"); break;
  default:  System.out.println ("drei Fragezeichen");
}
```

```
char   c = 'a';
String text = "";
while (c <= 'm')
{ text += c;
  c++;
}
System.out.println (text);

for (c = 'z', text = ""; c >= 'n'; c--) text += c;
System.out.println (text);

float  f[] = {1.2f, 2.3f, 7.8f, 8.9f};
for (float wert: f) System.out.println(wert);

int    i;
i = 100;
do
{
  i = i / 2;
  System.out.println("i = " + i);
} while ( i > 0 );
```

```
abcdefghijklm
zyxwvutsrqpon
1.2
2.3
7.8
8.9
i = 50
i = 25
i = 12
i = 6
i = 3
i = 0
```

Ausnahmebehandlung

was passiert wenn:

```
stringToConvert = "1A";// "1"
int nummer = Integer.parseInt( stringToConvert );
```

```
java.lang.NumberFormatException: For input string: "1A"
at java.lang.NumberFormatException.forInputString(Unknown Source)
at java.lang.Integer.parseInt(Unknown Source)
at java.lang.Integer.parseInt(Unknown Source)
at StrMan.main(StrMan.java:64)
```

Ausnahme (Exception) – ein nicht geplantes Ereignis, das zum Programmabsturz führt

Ausnahmebehandlung

wie läuft das ab:

```
stringToConvert = "1A";
try{
  nummer = Integer.parseInt( stringToConvert );
}
catch ( NumberFormatException e ){
   System.err.printf( "'%s' kann man nicht in eine
            Zahl konvertieren!%n", stringToConvert );
   nummer=0;
}
System.out.println( "Weiter geht's" );
```

wäre nicht if-Anweisung einfacher?

Ausnahmebehandlung - Trennung zwischen dem Sollverhalten und der Ausnahmesituation

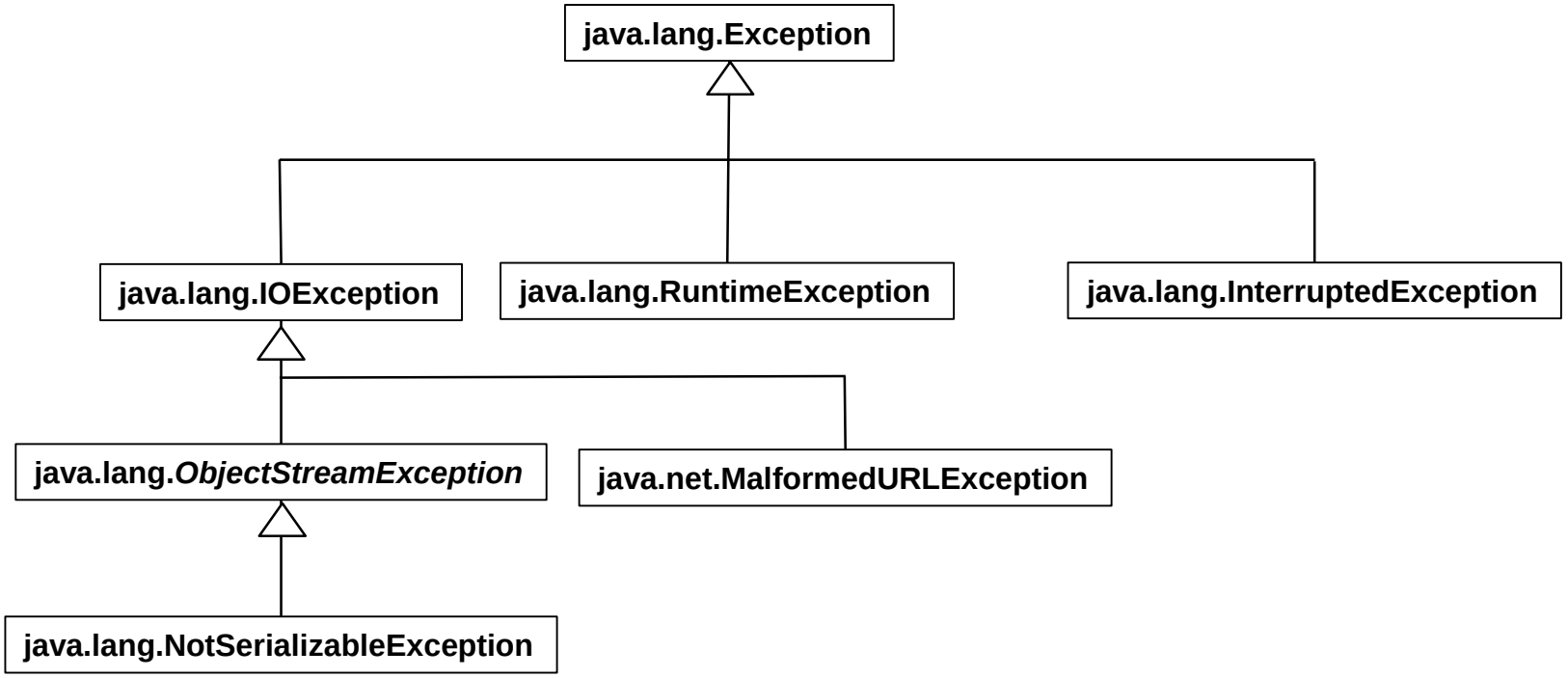
try-catch-Anweisung

```
catch (Exception  e){
}
geht immer!
```

```
try { Anweisungen }
catch ( Ausnahme oa1 ) {Anweisungen bei Ausnahme oa1}
catch ( Ausnahme oa2 ) {Anweisungen bei Ausnahme oa2}
:
catch ( Ausnahme oan ) {Anweisungen bei Ausnahme oan}
[finally { Anweisungen als abschließende Maßnahmen }]
```

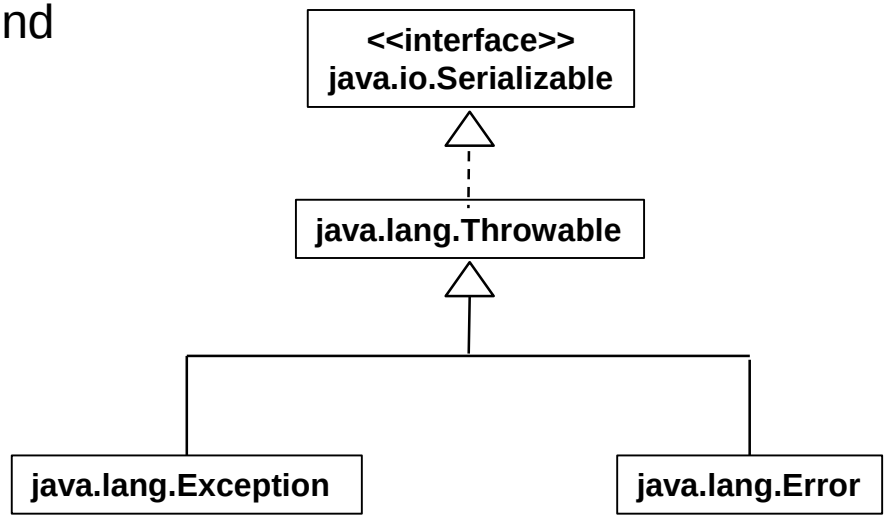
finally

- dient zum Durchführen abschließenden Maßnahmen, z.B. Objekte sichern, Dateien schließen
- wird immer ausgeführt, in allen 3 Fällen:
  - Es gab keine Ausnahme!
  - Es gab eine Ausnahme und diese wurde in einer catch-Anweisung behandelt!
  - Es gab eine Ausnahme, aber keine catch-Anweisung war für diesen Fehler zuständig!


java.lang.Exception
java.lang.IOException
java.lang.RuntimeException
java.lang.InterruptedException
java.lang.*ObjectStreamException*
java.net.MalformedURLException
java.lang.NotSerializableException

Exception – Basisklasse für alle Exception

Throwable – Basisklasse für Objekte, die Ausnahmesituationen (Errors und Exception) signalisieren



Throwable-Methoden:

```
String getMessage()
void printStackTrace()
```

was können wir dem Exception-Objekt entnehmen?

```
String name  = e.getClass().getName();
// z.B. java.lang.NumberFormatException

String msg   = e.getMessage();
// z.B. For input string: "1A"

String toStr = e.toString();
// z.B. java.lang.NumberFormatException: For input string: "1A"
```

Ist die Ausnahmenbehandlung immer sinnvoll? und

was ist eine Ausnahme und was ist ein Fehler?

`Error` – nicht reparierbar Laufzeitfehler oder Hardware-Problem
`java.lang.Error` schlecht behandelbar

`Exception` – kein Fehler, unerwarteter Fall zur Laufzeit
`IOException, InterruptedException` sinnvoll behandelbar

`RuntimeException` – möglicherweise Programmierfehler

RuntimeException-Beispiele:
- Fehler beim Parsen (`NumberFormatException`)
- Division durch 0 (`ArithmeticException`)
- Grenzüberschreitung bei Array (`IndexOutOfBoundsException`)
- Objekt existiert nicht (`NullPointerException`)

```
public class Ausnahmen {
//…
public void methode(Klasse objekt) {
  try{
       Point p = objekt.getPoint();
          //objekt kann null sein
          //bzw. null liefern
          //wäre aber auch mit if-Anweisung auszuschließen
       p.setX(20);
       //…
  }
  catch(java.lang.NullPointerException e){
       System.err.println(e);
  }
 }
}
```

Aber muss die Methode alles selbst behandeln?

Nein, sie kann Exception an die aufzurufende Methode weiter leiten!

```
public class Ausnahmen {
//…
public void methode(Klasse objekt)
                throws NullPointerException {
        Point p = objekt.getPoint();
        p.setX(20);
        //…
   }
}

public static void main(String args []){
    try{
         new Ausnahmen().methode();
    }
    catch(java.lang.NullPointerException e){
         System.err.println(e);
    }
}
```

Ausnahmeobjekt „manuell“ auswerfen: `throw objekt`

```
public class AusnahmeRaus {
public void methode() throws NullPointerException{
        //…
        throw new NullPointerException();
   }
}

public class AusnahmeRein {
public static void main(String[] args) {
        AusnahmeRaus o = new AusnahmeRaus();
        try{
                o.methode();
        }
        catch(NullPointerException e){
                e.printStackTrace(System.err);
        }
   }
}
```

Und wenn es keine passende Exception gibt?

Jeder Zeit ist die Definition eigener Exception möglich, abgeleitet von Exception oder einer anderen passenden Klasse

```
public class EinfachLeer extends Exception{
 //…
}

public class AusnahmeRaus {
       private String beutel;
       public void methode() throws EinfachLeer {
         if (beutel.isEmpty()) throw new EinfachLeer();
         //…
       }
}
```