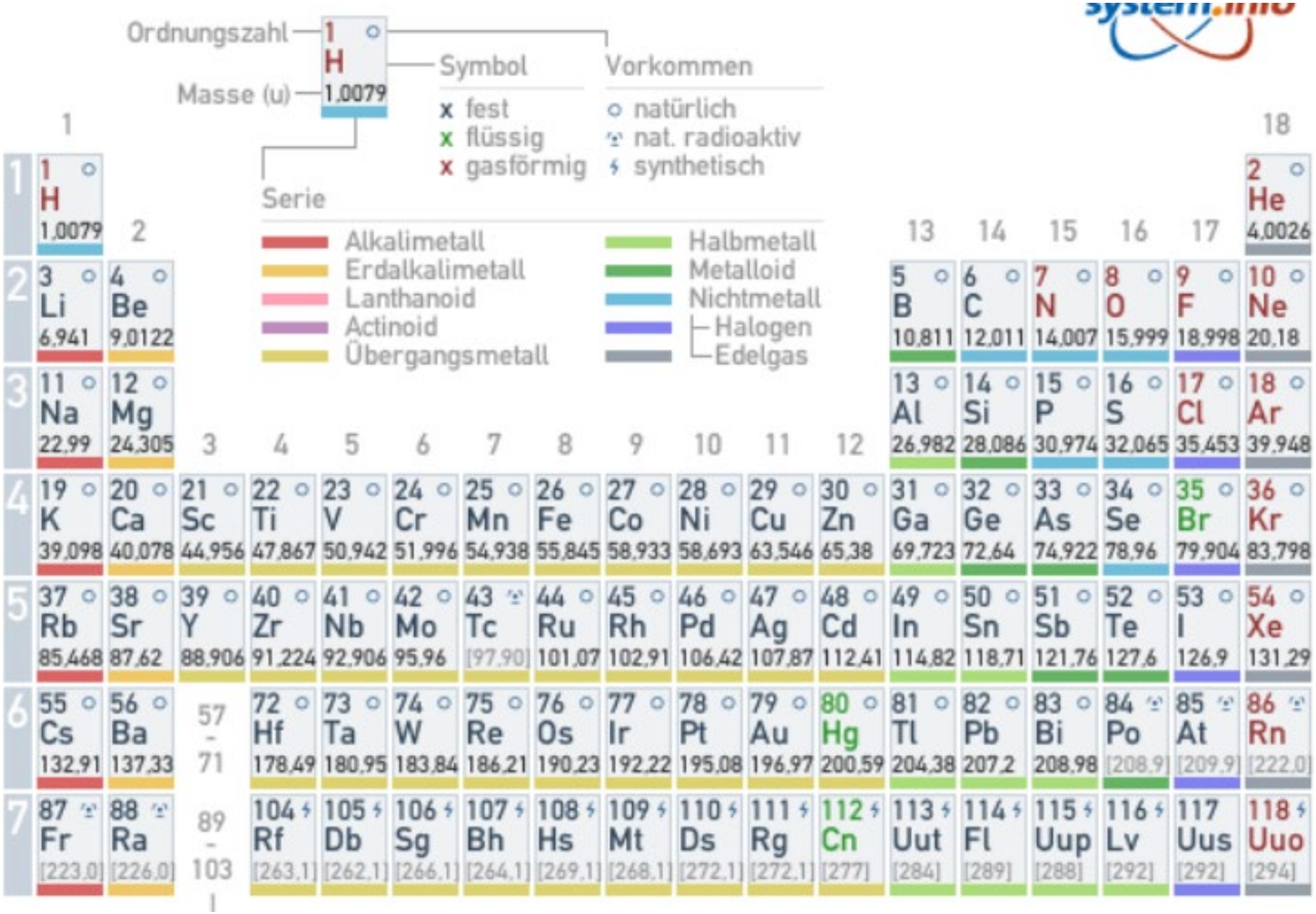
Lektion 11

Mehrdimensionale Arrays
Überladen von Methoden
Exceptions I
(optional) Bitweise Operatoren
Referenzen als Argumente

Wie bildet man die Struktur des Periodensystems der Elemente in einem Programm ab?



source: http://www.periodensystem.info/periodensystem/

Mehrdimensionale Arrays

```
float[][] matrix = new float[2][3];
matrix[0][0] = 1.1f;
matrix[0][1] = 1.2f;
matrix[0][2] = 1.3f;
matrix[1][0] = 2.1f;
matrix[1][1] = 2.2f;
matrix[1][2] = 2.3f;
for (int i = 0; i < matrix.length; i++)
{
  for (int j = 0; j < matrix[i].length; j++)
  {
    System.out.print(matrix[i][j] + " ");
  }
  System.out.println();
}
```

definiere Array von Arrays
Array beinhaltet 2 Arrays mit je 3 Elementen

1.1	1.2	1.3
2.1	2.2	2.3

Mehrdimensionale Arrays

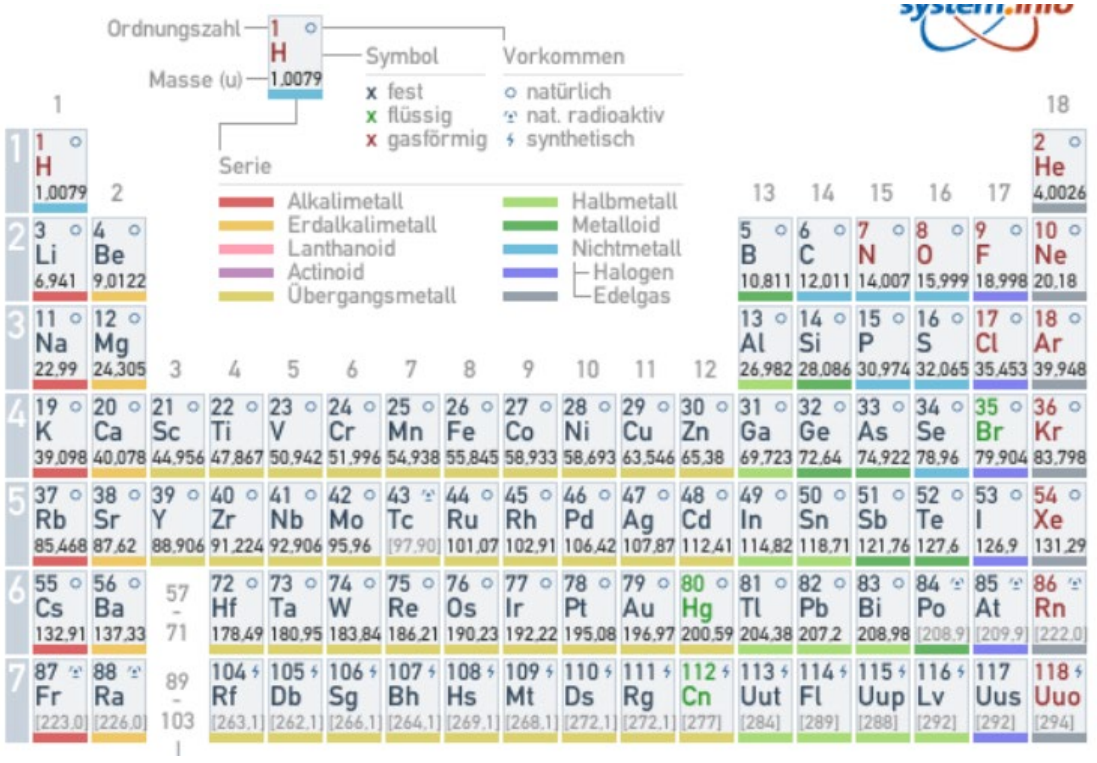
- Alternativ kann die Initialisierung auch in der Deklarationszeile erfolgen:

```
float[][] matrix = new float[][] {{1.1f,  1.2f,  1.3f}, {2.1f,  2.2f, 2.3f}};
```

- oder wieder in Kurzform:

```
float[][] matrix = {{1.1f,  1.2f,  1.3f}, {2.1f,  2.2f, 2.3f}};
```

Die Anzahl der Elemente in jeder Reihe kann von Reihe zu Reihe variieren (wie beim Periodensystem der Elemente).



source: http://www.periodensystem.info/periodensystem/

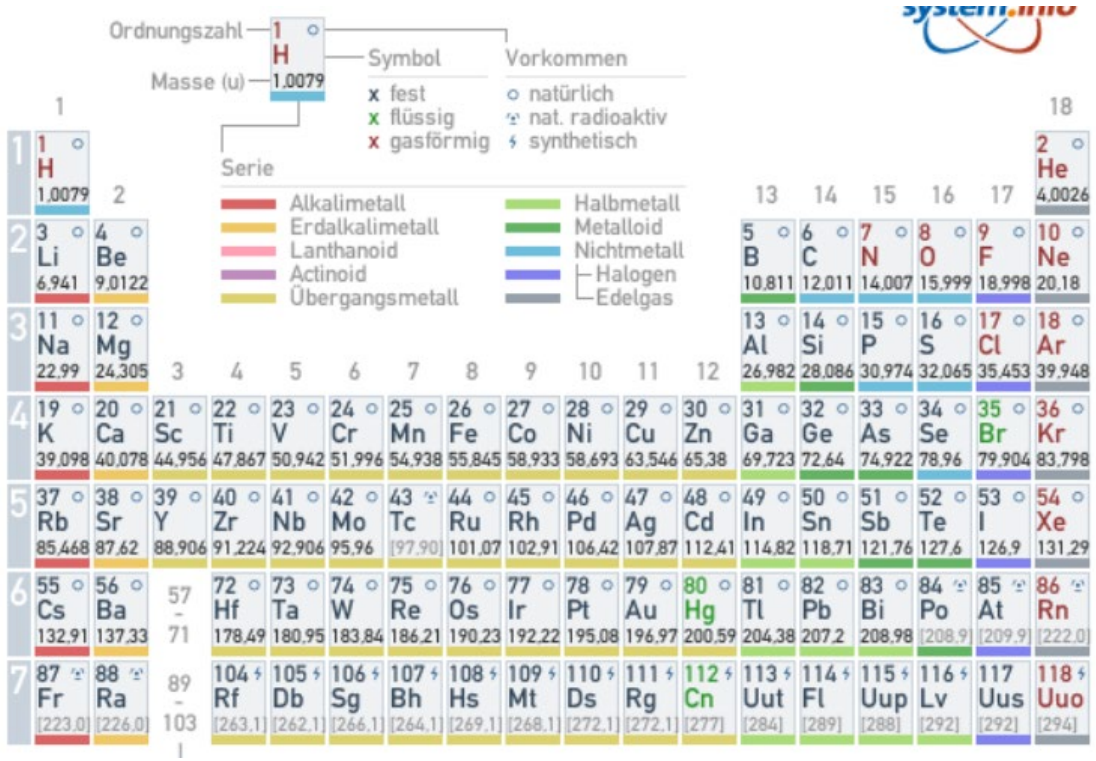
Mehrdimensionale Arrays
Bsp.: Periodensystem der Elemente

```
public class Element {
  String name;
  String kuerzel;
  int ordnungszahl;
  String gruppe;
  double masse;

 public Element(String name, String kuerzel, int ordnungszahl, String gruppe)
 {
    this.name = name;
    this.kuerzel = kuerzel;
    this.ordnungszahl = ordnungszahl;
    this.gruppe = gruppe;
  }
}
```

source:
http://www.periodensystem.info/periodensystem/

Mehrdimensionale Arrays
Bsp.: Periodensystem der Elemente

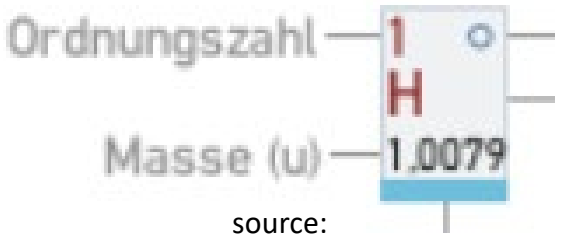
```
public static void main(String[] args) {
  Element[][] ps = new Element[7][]; //Deklariert ein Array von 7 (Referenzen auf) Arrays
  ps[0] = new Element[2]; //das erste Array hat 2 Elemente
  ps[1] = new Element[8]; //das zweite Array hat 8 Elemente
  ps[2] = new Element[8];
  ps[3] = new Element[18];
  ps[4] = new Element[18];




}
```



source: http://www.periodensystem.info/periodensystem/

Mehrdimensionale Arrays
Bsp.: Periodensystem der Elemente

```
public static void main(String[] args) {
  Element[][] ps = new Element[7][]; //Deklariert ein Array von 7 Arrays
  ps[0] = new Element[2]; //das erste Array hat 2 Elemente
  ps[1] = new Element[8]; //das zweite Array hat 8 Elemente
  ps[2] = new Element[8];
  ps[3] = new Element[18];
  ps[4] = new Element[18];
  …
  ps[0][0] = new Element("Wasserstoff", "H", 1, "1");
  ps[0][1] = new Element("Helium", "He", 2, "8");
  ps[1][0] = new Element("Lithium", "Li", 3, "1");
  ps[1][1] = new Element("Beryllium", "Be", 4, "2");
  ps[1][2] = new Element("Bor", "B", 5, "3");
  …
}
```



source:
http://www.periodensystem.info/periodensystem/

Modifier final

- Durch `final` erzwingt man, dass auf ein Klassenattribut, Instanzattribut od. eine lokale Variable nur **Lesezugriffe** erlaubt sind.
- `final` bietet sich bspw. für Werte an, die wiederholt im Quellcode auftauchen, z. B. Mehrwertsteuersatz, Zählerobergrenze, etc.

➢ Änderungen im Programm können dann an einer einzelnen Stelle durchgeführt werden.

- Die Variable wird dann i. d. R. (symbolische) Konstante genannt und in Großbuchstaben geschrieben.

```
public class VATExample {
  public static final float VAT_GERMANY = 19.0f;
  public static void main(String[] args) {
    int amount = 10000;
    System.out.println(amount * VAT_GERMANY/100.0);
  }
}
```

19.0 kann in verschiedenen Zusammenhänge für etwas Unterschiedliches stehen. VAT_GERMANY ist eindeutiger.

Überladen von Methoden

- Eine ***überladene Methode*** ist eine Methode mit dem **gleichen Namen** wie eine andere Methode in derselben Klasse – aber einer **unterschiedlichen Parameterliste**.
- Die Parameterliste kann sich unterscheiden durch
  - unterschiedliche Typen der Parameter
  - unterschiedliche Anzahl von Parametern

Überladen von Methoden:
unterschiedliche Typen der Parameter

- Die `print()`- und `println()`-Methoden von der Klasse `PrintStream` sind überladen.
- Das statische Objekt `out` der Klasse `System` ist bspw. vom Typ `PrintStream`:

```
public class PrintStream {
  public void println(boolean b) {
    …
  }
  public void println(char c) {
    …
  }
  public void println(int i) {
    …
  }
  …
}
```

Überladen von Methoden:
unterschiedliche Anzahl von Parametern

- Methode `calculateVAT()` wird zweimal implementiert:
  - Variante 1 mit festen Mehrwertsteuersatz
  - Variante 2 mit einem variablen Mehrwertsteuersatz

```
public class Ueberladen {
  public static final float VAT_GERMANY = 19.0f;
  public static double calculateVAT(double amount, double vat) {
    return amount * (vat/100.0);
  }
  public static double calculateVAT(double amount) {
    return calculateVAT(amount, VAT_GERMANY);
  }
  public static void main(String[] args) {
    System.out.println(calculateVAT(100));
    System.out.println(calculateVAT(100, 21.0));
  }
}
```

Exceptions (Teil I)

Wie gehen wir mit Fehlern um?

Beispiel: Umwandlung eines Strings in eine Zahl

```
String test = "25A";
int a = Integer.valueOf(test);
```

Wenn sich im String **test** eine Zahl befindet, wird diese in einen Integer umgewandelt. Was passiert bei Buchstaben?

Wir müssen dem Aufrufer eine Fehlermeldung mitteilen!

Warum benutzt man nicht einfach den Rückgabewert einer Methode, wenn ein Fehler auftritt, bspw. -1?

Der Rückgabewert der Methode (vom Typ int) benötigt bereits das komplette Intervall von -2^{31} bis $2^{31}-1$.

Daher ist ein Mechanismus sinnvoll, der auch ohne Verwendung des Rückgabewerts angeben kann, ob ein Fehler aufgetreten ist, nämlich Exceptions (Ausnahmen).

```
public static void main(String[] args)
{
  Scanner scanner = new Scanner(System.in);
  System.out.println("Bitte geben Sie eine Zahl ein: ");
  int zahl = scanner.nextInt();
  System.out.println(zahl);
  scanner.close();
}
```

Was passiert bei der Eingabe von Buchstaben?

```
Bitte geben Sie eine Zahl ein:
ew
Exception in thread "main" java.util.InputMismatchException
at java.util.Scanner.throwFor(Scanner.java:864)
at java.util.Scanner.next(Scanner.java:1485)
at java.util.Scanner.nextInt(Scanner.java:2117)
at java.util.Scanner.nextInt(Scanner.java:2076)
at exceptions.ScannerExceptionTest.main(ScannerExceptionTest.java:11)
```

Wir haben eine Exception ausgelöst!

Was passiert beim Auslösen einer Exception?

Das Laufzeitsystem unterbricht den normalen Ablauf.

Es generiert ein Exception-Objekt, in dem es genau festhält, wann, wo und was schief gegangen ist.

Dieses Exception-Objekt wird „geworfen“ in der Hoffnung, dass es jemand auffängt und eine Lösung parat hat.

Falls es niemand fängt, beendet das Laufzeitsystem das Programm.

Das Auslösen einer Exception nennt man auch „Werfen“ einer Exception.

Wie können wir eine geworfene Exception fangen?

Mit einem try-catch Block bestehend aus

- dem Schlüsselwort **try** gefolgt von einem Block
- dem Schlüsselwort **catch** gefolgt vom Exceptionnamen, gefolgt von einem Block

```
try
{
  //Anweisungen, die wir versuchen wollen, auszuführen.
}
catch(<class name of exception> e)
{
  //Anweisungen, die wir ausführen, wenn im try-Block ein Fehler auftritt
}
```

Wie können wir eine geworfene Exception fangen?

```
Scanner scanner = new Scanner(System.in);
System.out.println("Bitte geben Sie eine Zahl ein: ");
try
{
  int zahl = scanner.nextInt();
  System.out.println(zahl);
}
catch(InputMismatchException e)
{
  System.out.println("Fehler beim Einlesen der Zahl!");
}
scanner.close();
```

Code, den wir versuchen auszuführen

Fehlerfall, den wir behandeln wollen

Code, den wir im Fehlerfall ausführen

Wie können wir den Fehler beheben?

```
Scanner scanner = new Scanner(System.in);
boolean fehlerGefunden;
do
{
  try
  {
    System.out.println("Bitte geben Sie eine Zahl ein: ");
    int zahl = scanner.nextInt();
    fehlerGefunden = false;
    System.out.println(zahl);
  }
  catch (InputMismatchException e)
  {
    System.out.println("Fehler beim Einlesen der Zahl!");
    scanner.nextLine();
    fehlerGefunden = true;
  }
}
while (fehlerGefunden);
scanner.close();
```

kein Fehler aufgetreten

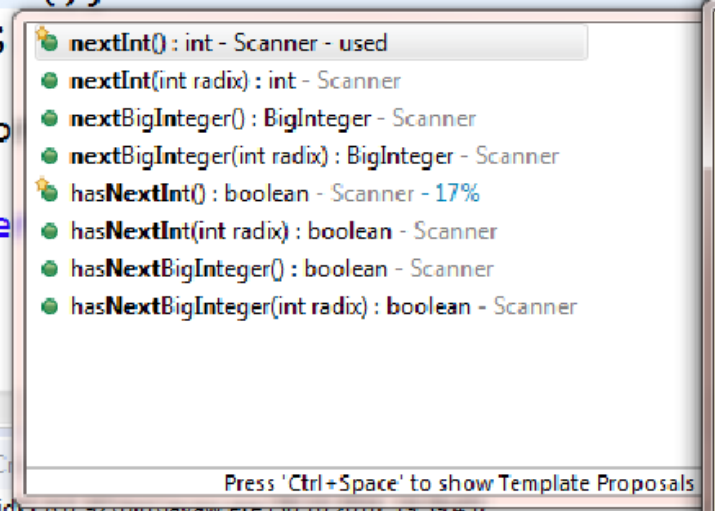
lies den fehlerhaften Input, ansonsten wirft nextInt() den gleichen Fehler, weil der Eingabepuffer unverändert ist.

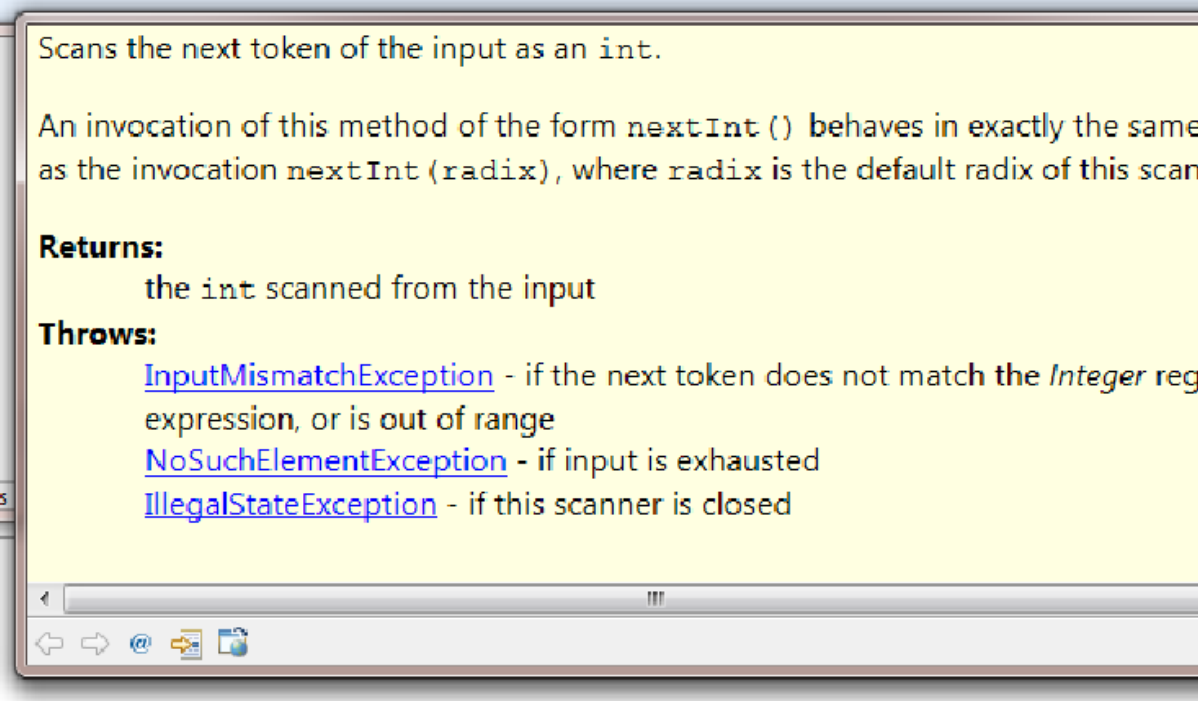
wiederhole solange ein Fehler auftritt

Woher wissen wir überhaupt, wo Fehler auftreten können?

Java Doc

```
Scanner scanner = new Scanner(System.in);
System.out.println("Bitte geben Sie eine Zahl ein: ");
try
{
  int zahl = scanner.nextInt();
  System.out.println(zahl);
}
catch(InputMismatchExceptio
{
  System.out.println("Fehle
}
scanner.close();
```





Javadoc | Declaration | Search | Console

nnerExceptionTest [Java Application] C:\Program Files\Java\

```
geben Sie eine Zahl ein:

beim Einlesen der Zahl!
```

Um uns das “Weglesen” der fehlerhaften Eingabe zu sparen, kann man auch direkt alle Eingaben als String lesen und in einen Integer umwandeln:

```
Scanner scanner = new Scanner(System.in);
System.out.println("Bitte geben Sie eine Zahl ein: ");
String eingabe = scanner.nextLine();
int zahl = Integer.valueOf(eingabe);
scanner.close();
```

Wenn in eingabe Nichtzahlen auftreten, löst Integer.valueOf eine Exception aus:

```
Bitte geben Sie eine Zahl ein:
afs
Exception in thread "main" java.lang.NumberFormatException: For input string: "afs"
at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
at java.lang.Integer.parseInt(Integer.java:580)
at java.lang.Integer.valueOf(Integer.java:766)
at exceptions.ScannerExceptionTest.testWithNextLineOnly(ScannerExceptionTest.java:75)
at exceptions.ScannerExceptionTest.main(ScannerExceptionTest.java:13)
```

```
Scanner scanner = new Scanner(System.in);
boolean fehlerGefunden;
do
{
  try
  {
    System.out.println("Bitte geben Sie eine Zahl ein: ");
    String eingabe = scanner.nextLine();
    int zahl = Integer.valueOf(eingabe);
    fehlerGefunden = false;
    System.out.println(zahl);
  }
  catch (NumberFormatException e)
  {
    System.out.println("Fehler beim Einlesen der Zahl!");
    fehlerGefunden = true;
  }
}
while (fehlerGefunden);
scanner.close();
```

```
public static int tageImMonat(String monat)
{
  int tage = switch(monat)
  {
    case "Februar" -> 28;
    case "April", "Juni", "September", "November" -> 30;
    case "Januar", "März", "Mai", "Juli", "August", "Oktober", "Dezember" -> 31;
    default ->
    {
      System.out.println("Falscher Monatsname");
      yield -1;
    }
  };
  return tage;
}
```

Was passiert,
wenn ein falscher Monat übergeben wird?

Die Methode sollte besser einen Fehler auslösen.

```
public static int tageImMonat(String monat)
{
  int tage = switch(monat)
  {
    case "Februar" -> 28;
    case "April", "Juni", "September", "November" -> 30;
    case "Januar", "März", "Mai", "Juli", "August", "Oktober", "Dezember" -> 31;
    default -> throw new RuntimeException("falscher Monatsname");
  };
  return tage;
}
```

Hier wird tatsächlich ein Fehler ausgelöst:

Es wird ein neues RuntimeException-Objekt angelegt
und über das **throw**-Schlüsselwort geworfen.

Bitweise Operatoren

& - Bitweises Und

| - Bitweises Oder

^ - Bitweises XOR (Exklusives Oder)

~ - Komplement

```
int a = 0b0110;
int b = 0b1100;
byte c = 0b00000110;
System.out.println(a & b);
System.out.println(a | b);
System.out.println(a ^ b);
System.out.println(~c);
```

```
  0110
& 1100
------
a & b -> 0100

  0110
| 1100
------
a | b -> 1110

  0110
^ 1100
------
a ^ b -> 1010

   ~ 00000110
   ----------
~c ->  11111001
```

Binärdarstellung: Negative Zahlen

```
System.out.println((byte)0b00000010);                    2
System.out.println((byte)0b00000001);                    1
System.out.println((byte)0b00000000);                    0
System.out.println((byte)0b11111111);                    -1
System.out.println((byte)0b11111110);                    -2
System.out.println((byte)0b11111101);                    -3
```

Verwendung von Bitweise Operatoren

```
public class EmbeddedSystemSimulator {
  public static final int RED    = 0b0001;
  public static final int GREEN  = 0b0010;
  public static final int YELLOW = 0b0100;
  public static final int BLUE   = 0b1000;
  public static final int ALL = RED | GREEN |
                          YELLOW | BLUE;
  int currentLights = 0;

  public static void main(String[] args) {
    EmbeddedSystemSimulator ess =
      new EmbeddedSystemSimulator();
    ess.turnOnLights(RED | YELLOW | GREEN);
    ess.turnOnLights(ALL);
    ess.turnOffLights(ALL);
    ess.turnOnLights(RED);
    ess.turnOnLights(YELLOW);
  }
  public void turnOnLights(int value) {
    currentLights = currentLights | value;
    printStatus();
  }
  public void turnOffLights(int value) {
    currentLights = currentLights & ~value;
    printStatus();
  }
  private void printStatus() {
    System.out.print("Folg. Lichter leuchten: ");
    if ((currentLights & RED) == RED)
      System.out.print("rot ");
    if ((currentLights & GREEN) == GREEN)
      System.out.print("grün ");
    if ((currentLights & YELLOW) == YELLOW)
      System.out.print("gelb ");
    if ((currentLights & BLUE) == BLUE)
      System.out.print("blau ");
    System.out.println();
  }
}
```

Auswertung von Ausdrücken

Operator(en)	Priorität
[ ] . (*<parameters>*) expr++ expr--	1 (höchste)
++expr --expr +expr -expr ~ ! (*<type>*) new	2
* / %	3
+ -	4
<< >> >>>	5
< <= > >= instanceof	6
== !=	7
&	8
^	9
|	10
&&	11
||	12
? :	13
= += -= *= /= %= <<= >>= >>>= &= ^= |=	14 (niedrigste)

arithmetic left shift operator
signed left shift operator

`<< x`

- schiebt ein Bitmuster um x Stellen nach **links**.
- entspricht einer Multiplikation mit 2^x
- Bits, die nach **links** aus dem Speicherbereich hinausgeschoben werden, gehen verloren.
- Von rechts wird mit Nullen aufgefüllt.

```
Bsp.:
5 << 2 wird ausgewertet zu 20
```

bzw.:

0000 0101 << 2 wird ausgewertet zu 0001 0100

arithmetic right shift operator
signed right shift operator

`>> x`

- schiebt ein Bitmuster um x Stellen nach **rechts**.
- entspricht einer ganzzahligen Division durch 2^x
- Bits, die nach **rechts** aus dem Speicherbereich hinausgeschoben werden, gehen verloren.
- erhält das Vorzeichen, d.h. wenn die vorderste Bitstelle eine 1 ist, werden Einser von links nachgeschoben, anders Nullen.

Bsp.:
`5 >> 1 wird ausgewertet zu 2`

bzw.:
0101 >> 1 wird ausgewertet zu 0010

logical right shift operator
unsigned right shift operator

>>> x

- entspricht >> x
- erhält aber das Vorzeichen nicht, d. h. es wird von links mit Nullen aufgefüllt.

Bitschiebe-Operatoren Beispiele

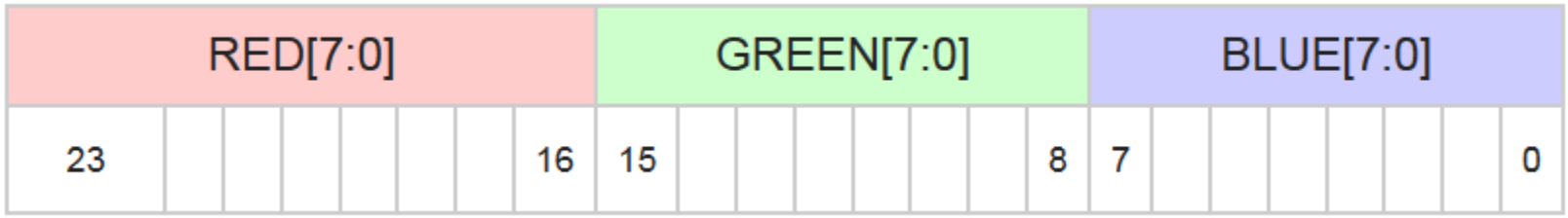
Binär	Dezimal	
00000000 00000000 00000000 00000001	1	
00000000 00000000 00000000 00000010	2	<< 1
00000000 00000000 00000000 00000100	4	<< 1
10000000 00000000 00000000 00000001	-2147483647	
00000000 00000000 00000000 00000010	2	<< 1
10000000 00000000 00000000 00000001	-2147483647	
11000000 00000000 00000000 00000000	-1073741824	>> 1
10000000 00000000 00000000 00000001	-2147483647	
01000000 00000000 00000000 00000000	1073741824	>>> 1

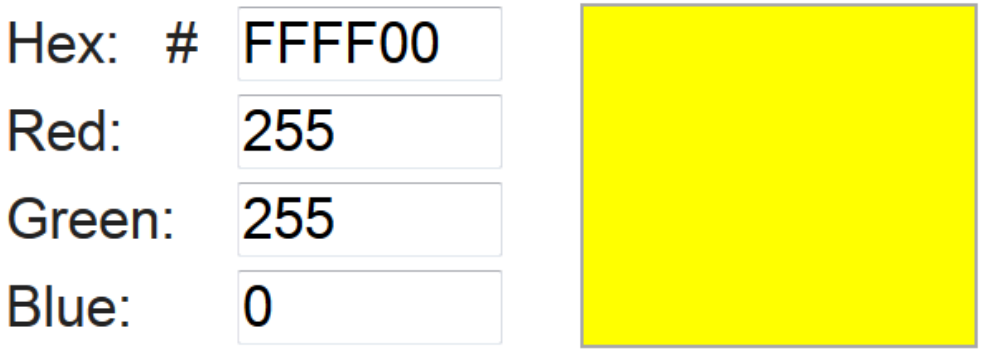
Bitschiebe-Operatoren

- Bitschiebeoperatoren arbeiten deutlich schneller als eine normale Multiplikation oder Division
- Compiler versuchen i. d. R. Multiplikationen oder Divisionen mit Bitschiebeoperatoren zu ersetzen, falls möglich.

Bitschiebe-Operatorenbeispiel aus der Computergrafik

- 24 Bit Farbpunkt im RGB-Modus kann z. B. wie folgt gespeichert werden:





Bitschiebe-Operatorenbeispiel aus der Computergrafik

```
int red = 255 << 16;
int green = 255 << 8;
int blue = 255;
System.out.printf("%06x\n", red);
System.out.printf("%06x\n", green);
System.out.printf("%06x\n", blue);
```

```
Ausgabe:
ff0000
00ff00
0000ff
```

%x – in Hexadezimal ausgeben
%6x – mit Leerzeichen von links auf 6 Stellen auffüllen
%06x – mit 0 von links auf 6 Stellen auffüllen

Auswertung von Ausdrücken

Operator(en)	Priorität
[ ] . (*<parameters>*) expr++ expr--	1 (höchste)
++expr --expr +expr -expr ~ ! (*<type>*) new	2
* / %	3
+ -	4
<< >> >>>	5
< <= > >= instanceof	6
== !=	7
&	8
^	9
|	10
&&	11
||	12
? :	13
= += -= *= /= %= <<= >>= >>>= &= ^= |=	14 (niedrigste)

Was passiert bei folgendem Code?

```
int yellow =  255 << 16  +  255 << 8;
System.out.printf("%06x\n", yellow);
```

Auswertung von Ausdrücken

Operator(en)	Priorität
[ ] . (*<parameters>*) expr++ expr--	1 (höchste)
++expr --expr +expr -expr ~ ! (*<type>*) new	2
* / %	3
+ -	4
<< >> >>>	5
< <= > >= instanceof	6
== !=	7
&	8
^	9
|	10
&&	11
||	12
? :	13
= += -= *= /= %= <<= >>= >>>= &= ^= |=	14 (niedrigste)

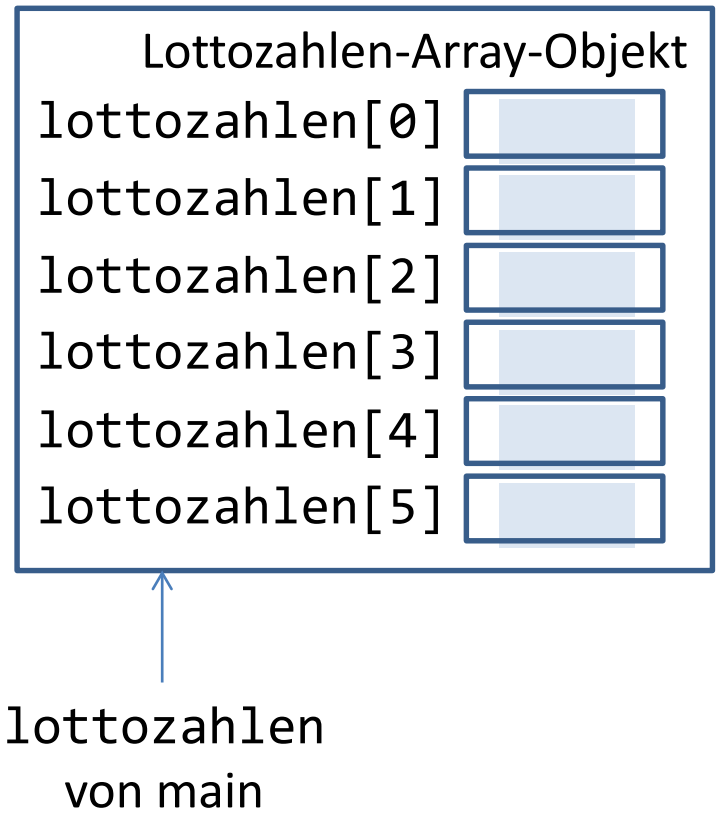
```
int yellow = (255 << 16) + (255 << 8);
System.out.printf("%06x\n", yellow);
```

Übergabe von Referenzen an Methoden

```
public class Arrays
{
  public static void initArray(int[] lottozahlen)
  {
    for (int i = 0; i < lottozahlen.length; i++)
    {
      lottozahlen[i] = (int)(Math.random()*49+1);
    }
  }

  public static void main(String[] args)
  {
    int[] lottozahlen = new int[6];
    initArray(lottozahlen);
    for (int i = 0; i < lottozahlen.length; i++)
    {
      System.out.print(lottozahlen[i] + " ");
    }
  }
}
```

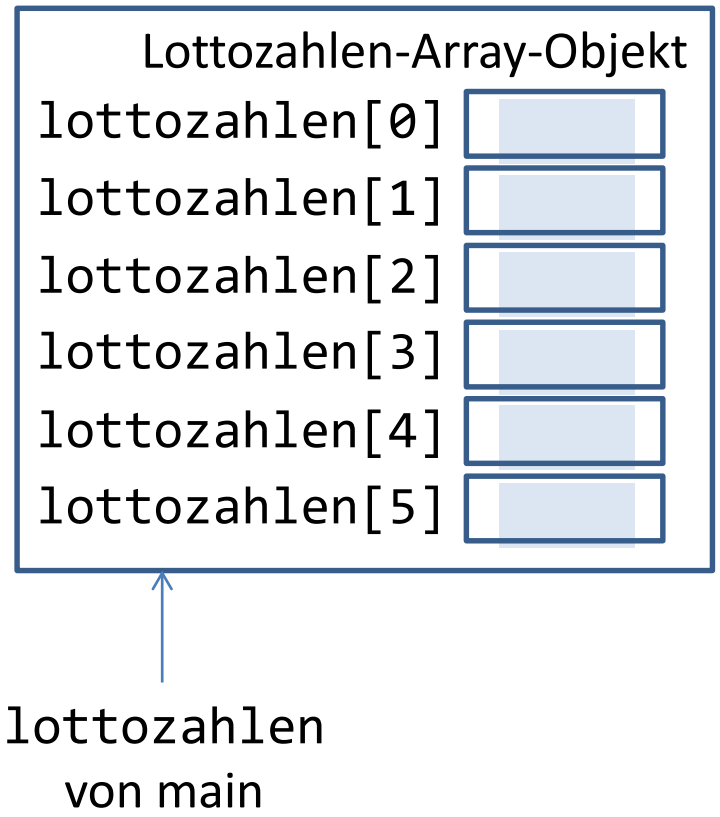
Übergabe von Referenzen an Methoden

```
public class Arrays
{
  public static void initArray(int[] lottozahlen)
  {
    for (int i = 0; i < lottozahlen.length; i++)
    {
      lottozahlen[i] = (int)(Math.random()*49+1);
    }
  }

  public static void main(String[] args)
  {
    int[] lottozahlen = new int[6];
    initArray(lottozahlen);
    for (int i = 0; i < lottozahlen.length; i++)
    {
      System.out.print(lottozahlen[i] + " ");
    }
  }
}
```



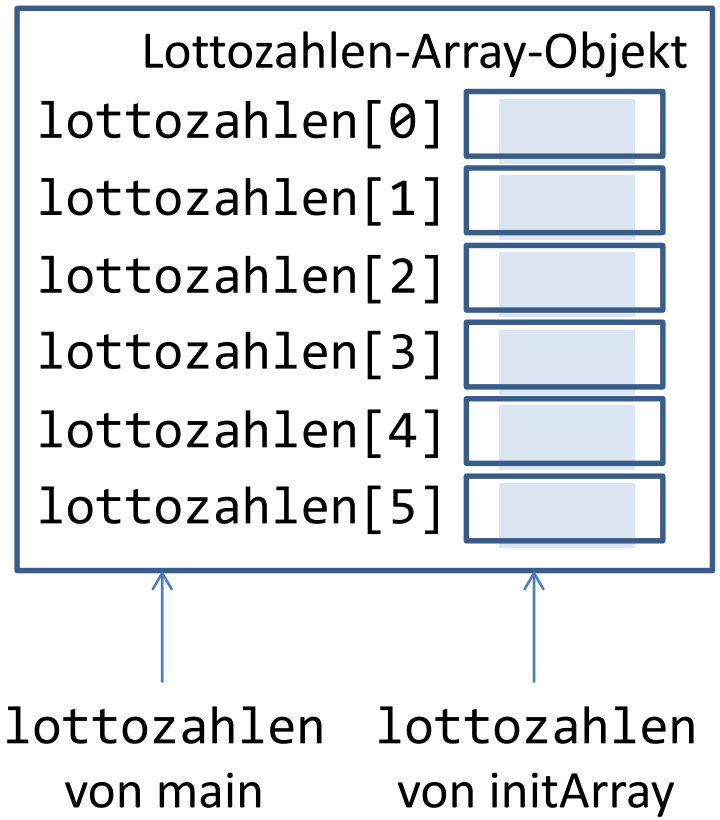
Übergabe von Referenzen an Methoden

```
public class Arrays
{
  public static void initArray(int[] lottozahlen)
  {
    for (int i = 0; i < lottozahlen.length; i++)
    {
      lottozahlen[i] = (int)(Math.random()*49+1);
    }
  }

  public static void main(String[] args)
  {
    int[] lottozahlen = new int[6];
    initArray(lottozahlen);
    for (int i = 0; i < lottozahlen.length; i++)
    {
      System.out.print(lottozahlen[i] + " ");
    }
  }
}
```



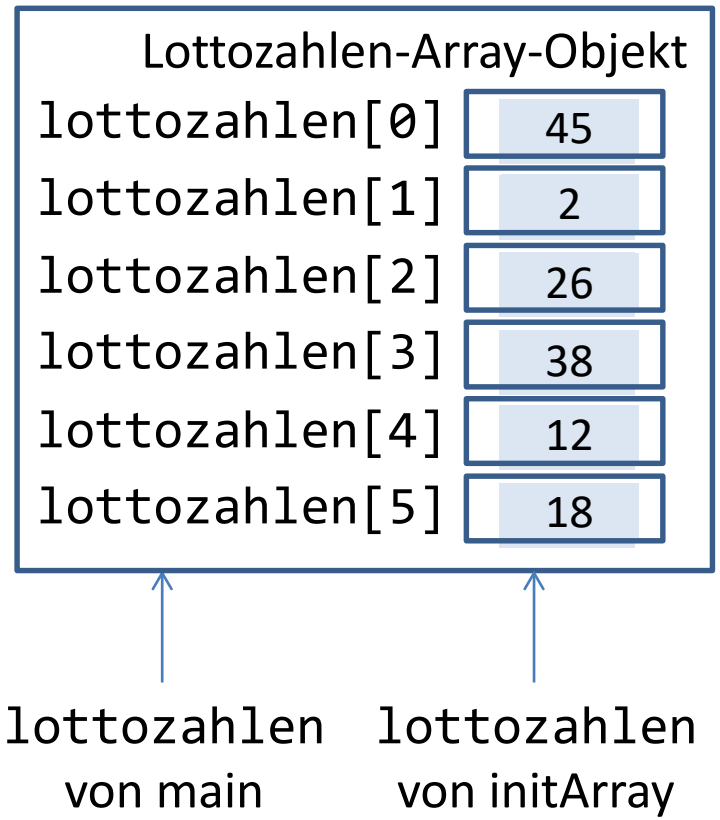
Übergabe von Referenzen an Methoden

```
public class Arrays
{
  public static void initArray(int[] lottozahlen)
  {
    for (int i = 0; i < lottozahlen.length; i++)
    {
      lottozahlen[i] = (int)(Math.random()*49+1);
    }
  }

  public static void main(String[] args)
  {
    int[] lottozahlen = new int[6];
    initArray(lottozahlen);
    for (int i = 0; i < lottozahlen.length; i++)
    {
      System.out.print(lottozahlen[i] + " ");
    }
  }
}
```



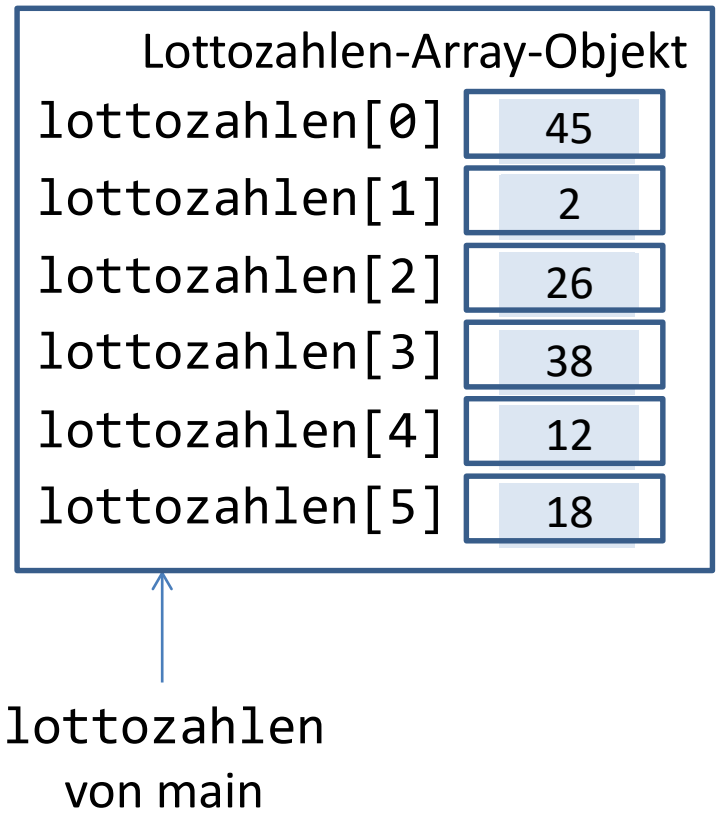
Übergabe von Referenzen an Methoden

```
public class Arrays
{
  public static void initArray(int[] lottozahlen)
  {
    for (int i = 0; i < lottozahlen.length; i++)
    {
      lottozahlen[i] = (int)(Math.random()*49+1);
    }
  }

  public static void main(String[] args)
  {
    int[] lottozahlen = new int[6];
    initArray(lottozahlen);
    for (int i = 0; i < lottozahlen.length; i++)
    {
      System.out.print(lottozahlen[i] + " ");
    }
  }
}
```



Übergabe von Referenzen an Methoden

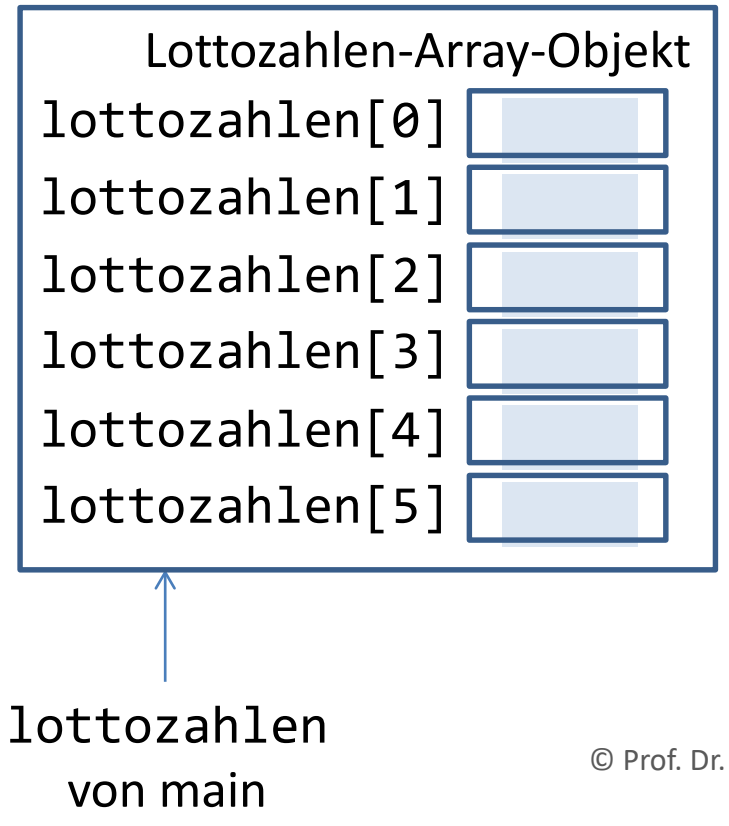
```
public class Arrays
{
  public static void initArray(int[] lottozahlen)
  {
    for (int i = 0; i < lottozahlen.length; i++)
    {
      lottozahlen[i] = (int)(Math.random()*49+1);
    }
  }

  public static void main(String[] args)
  {
    int[] lottozahlen = new int[6];
    initArray(lottozahlen);
    for (int i = 0; i < lottozahlen.length; i++)
    {
      System.out.print(lottozahlen[i] + " ");
    }
  }
}
```



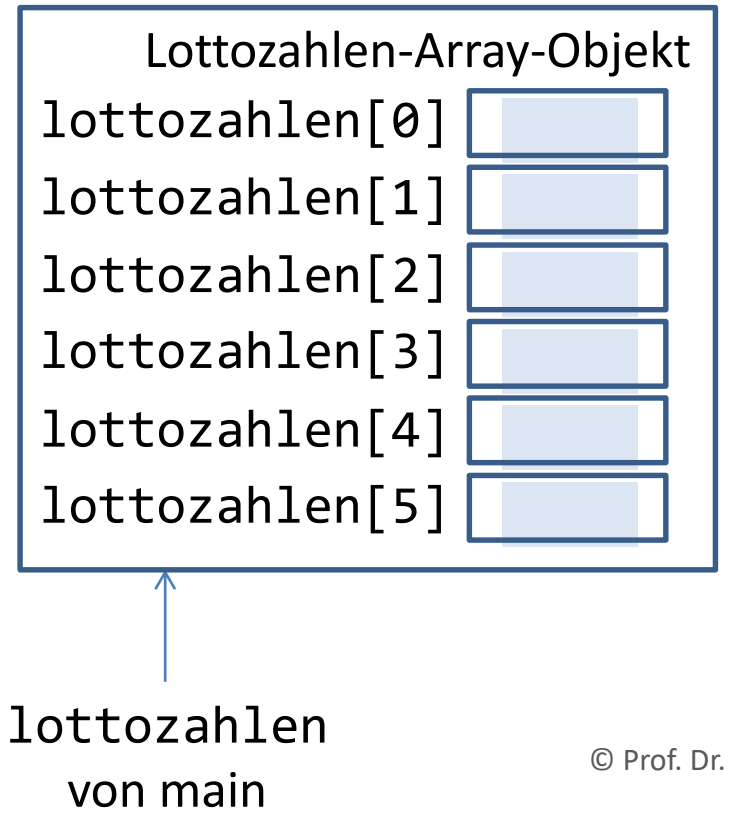
Was passiert bei folgendem Code?

```
public class Arrays
{
  public static void initArray2(int[] lottozahlen)
  {
    int[] zahlen = new int[6];
    for (int i = 0; i < zahlen.length; i++) {
      zahlen[i] = (int)(Math.random()*49+1);
    }
    lottozahlen = zahlen;
  }
  public static void main(String[] args)
  {
    int[] lottozahlen = new int[6];
    initArray2(lottozahlen);
    for (int i = 0; i < lottozahlen.length; i++)
    {
      System.out.print(lottozahlen[i] + " ");
    }
  }
}
```

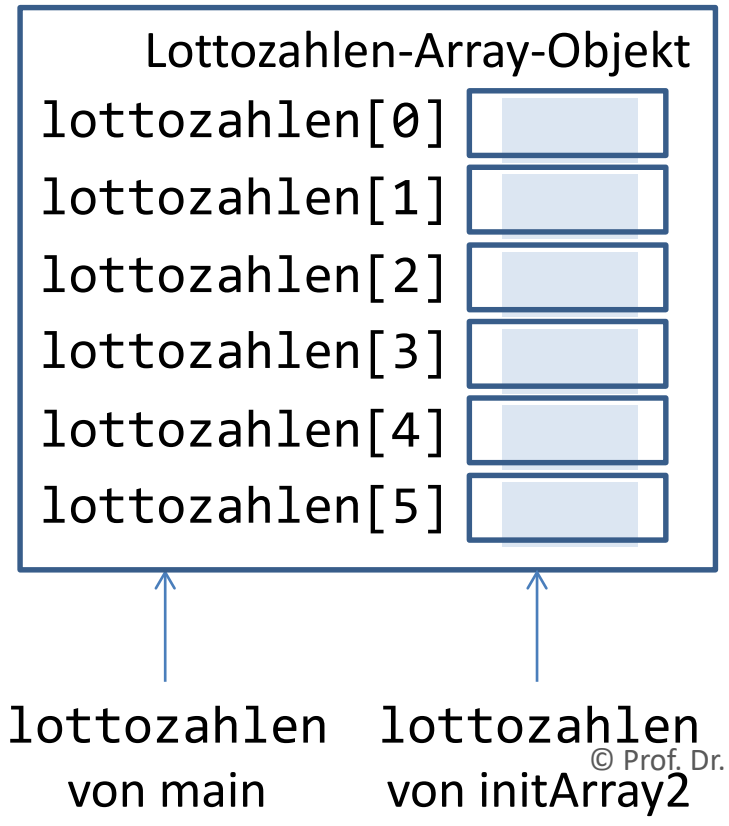
```
public class Arrays
{
  public static void initArray2(int[] lottozahlen)
  {
    int[] zahlen = new int[6];
    for (int i = 0; i < zahlen.length; i++) {
      zahlen[i] = (int)(Math.random()*49+1);
    }
    lottozahlen = zahlen;
  }
  public static void main(String[] args)
  {
    int[] lottozahlen = new int[6];
    initArray2(lottozahlen);
    for (int i = 0; i < lottozahlen.length; i++)
    {
      System.out.print(lottozahlen[i] + " ");
    }
  }
}
```



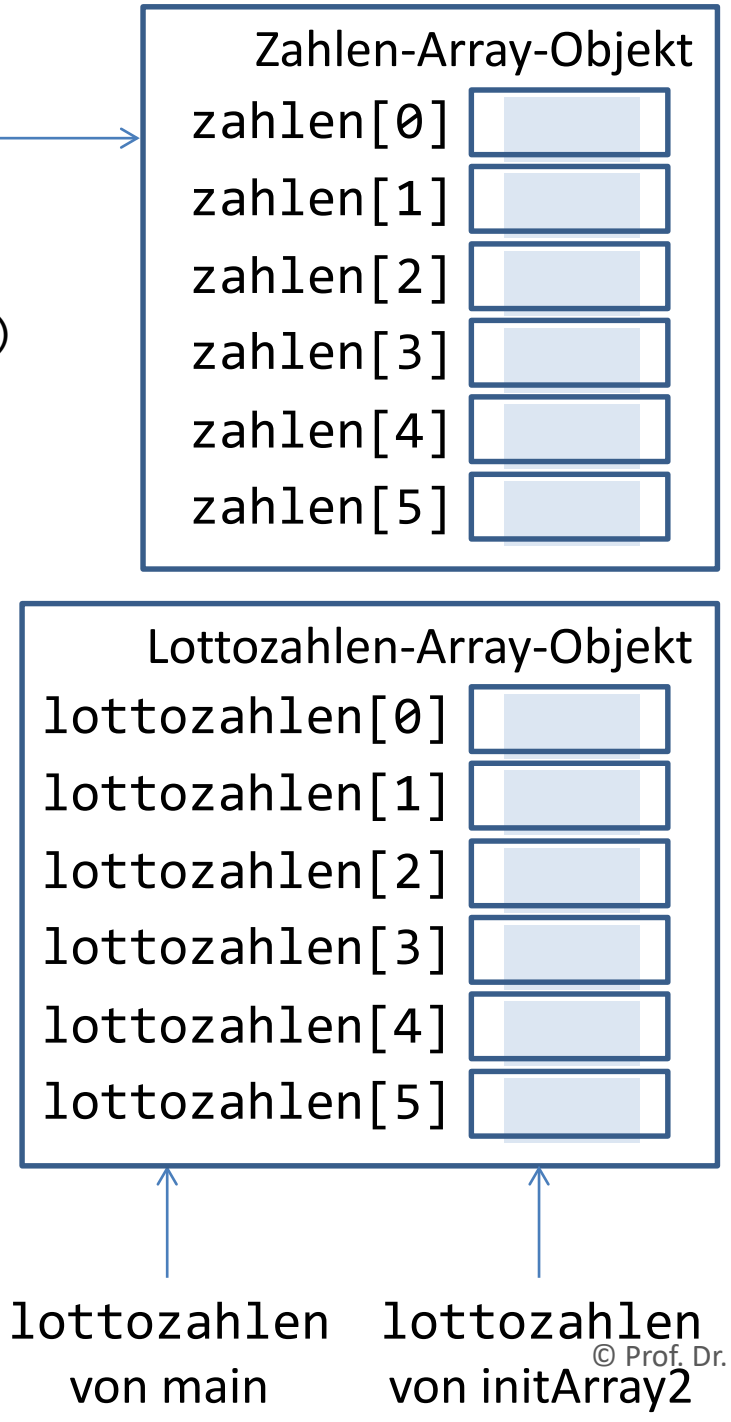
```
public class Arrays
{
  public static void initArray2(int[] lottozahlen)
  {
    int[] zahlen = new int[6];
    for (int i = 0; i < zahlen.length; i++) {
      zahlen[i] = (int)(Math.random()*49+1);
    }
    lottozahlen = zahlen;
  }
  public static void main(String[] args)
  {
    int[] lottozahlen = new int[6];
    initArray2(lottozahlen);
    for (int i = 0; i < lottozahlen.length; i++)
    {
      System.out.print(lottozahlen[i] + " ");
    }
  }
}
```



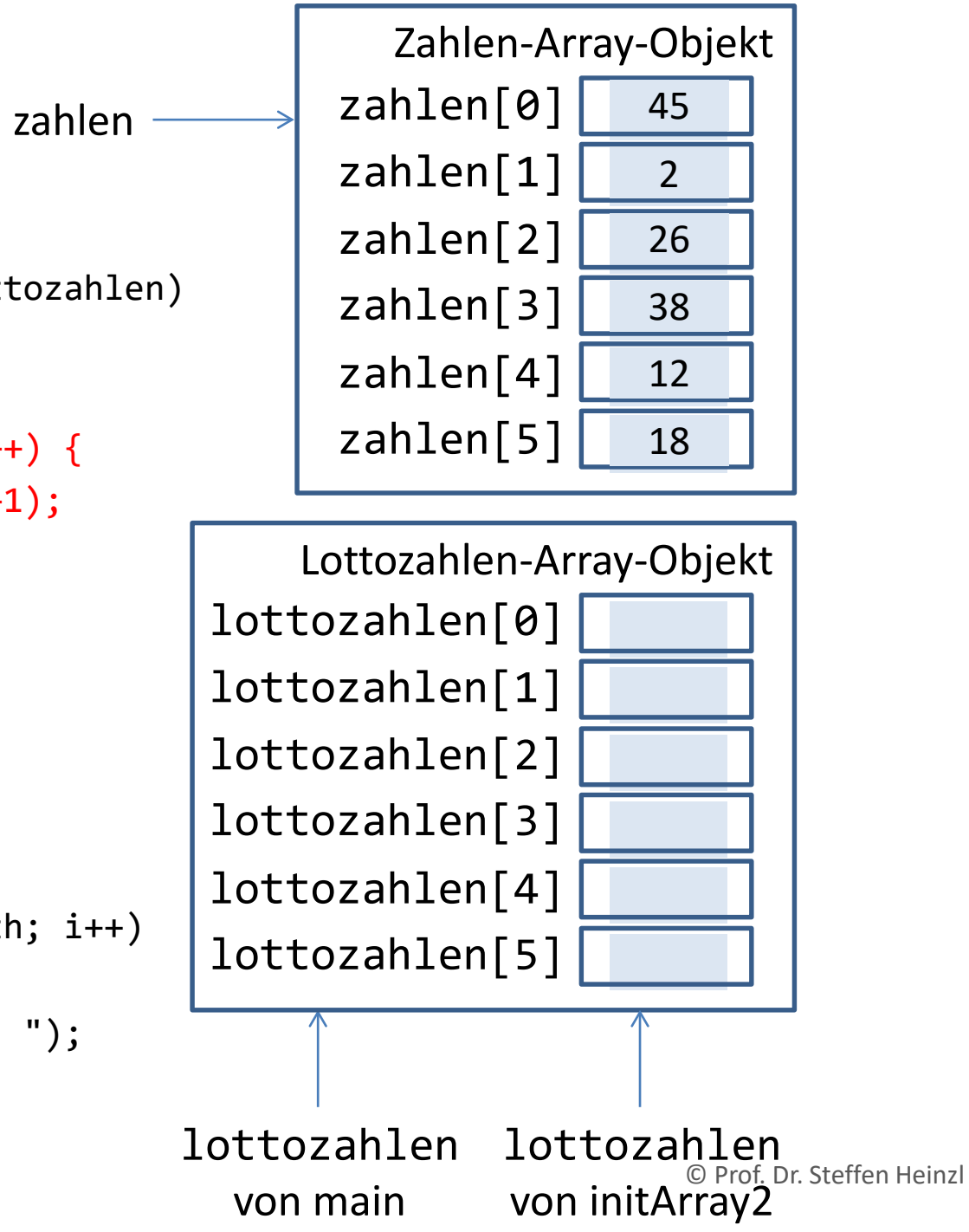
```
public class Arrays
{
  public static void initArray2(int[] lottozahlen)
  {
    int[] zahlen = new int[6];
    for (int i = 0; i < zahlen.length; i++) {
      zahlen[i] = (int)(Math.random()*49+1);
    }
    lottozahlen = zahlen;
  }
  public static void main(String[] args)
  {
    int[] lottozahlen = new int[6];
    initArray2(lottozahlen);
    for (int i = 0; i < lottozahlen.length; i++)
    {
      System.out.print(lottozahlen[i] + " ");
    }
  }
}
```



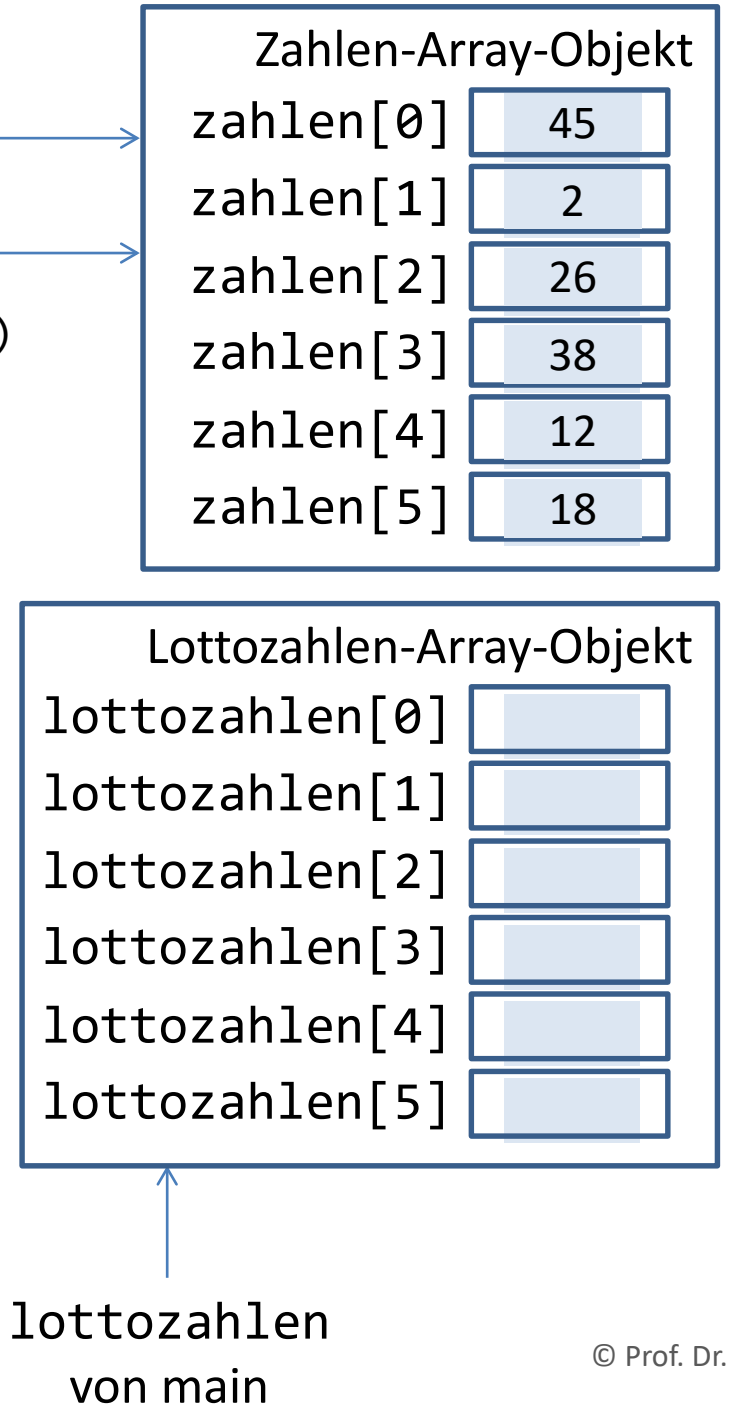
```
public class Arrays
{
  public static void initArray2(int[] lottozahlen)
  {
    int[] zahlen = new int[6];
    for (int i = 0; i < zahlen.length; i++) {
      zahlen[i] = (int)(Math.random()*49+1);
    }
    lottozahlen = zahlen;
  }
  public static void main(String[] args)
  {
    int[] lottozahlen = new int[6];
    initArray2(lottozahlen);
    for (int i = 0; i < lottozahlen.length; i++)
    {
      System.out.print(lottozahlen[i] + " ");
    }
  }
}
```



```
public class Arrays
{
  public static void initArray2(int[] lottozahlen)
  {
    int[] zahlen = new int[6];
    for (int i = 0; i < zahlen.length; i++) {
      zahlen[i] = (int)(Math.random()*49+1);
    }
    lottozahlen = zahlen;
  }
  public static void main(String[] args)
  {
    int[] lottozahlen = new int[6];
    initArray2(lottozahlen);
    for (int i = 0; i < lottozahlen.length; i++)
    {
      System.out.print(lottozahlen[i] + " ");
    }
  }
}
```



```
public class Arrays
{
  public static void initArray2(int[] lottozahlen)
  {
    int[] zahlen = new int[6];
    for (int i = 0; i < zahlen.length; i++) {
      zahlen[i] = (int)(Math.random()*49+1);
    }
    lottozahlen = zahlen;
  }
  public static void main(String[] args)
  {
    int[] lottozahlen = new int[6];
    initArray2(lottozahlen);
    for (int i = 0; i < lottozahlen.length; i++)
    {
      System.out.print(lottozahlen[i] + " ");
    }
  }
}
```



```
public class Arrays
{
  public static void initArray2(int[] lottozahlen)
  {
    int[] zahlen = new int[6];
    for (int i = 0; i < zahlen.length; i++) {
      zahlen[i] = (int)(Math.random()*49+1);
    }
    lottozahlen = zahlen;
  }
  public static void main(String[] args)
  {
    int[] lottozahlen = new int[6];
    initArray2(lottozahlen);
    for (int i = 0; i < lottozahlen.length; i++)
    {
      System.out.print(lottozahlen[i] + " ");
    }
  }
}
```