Cheatsheet M319: Java

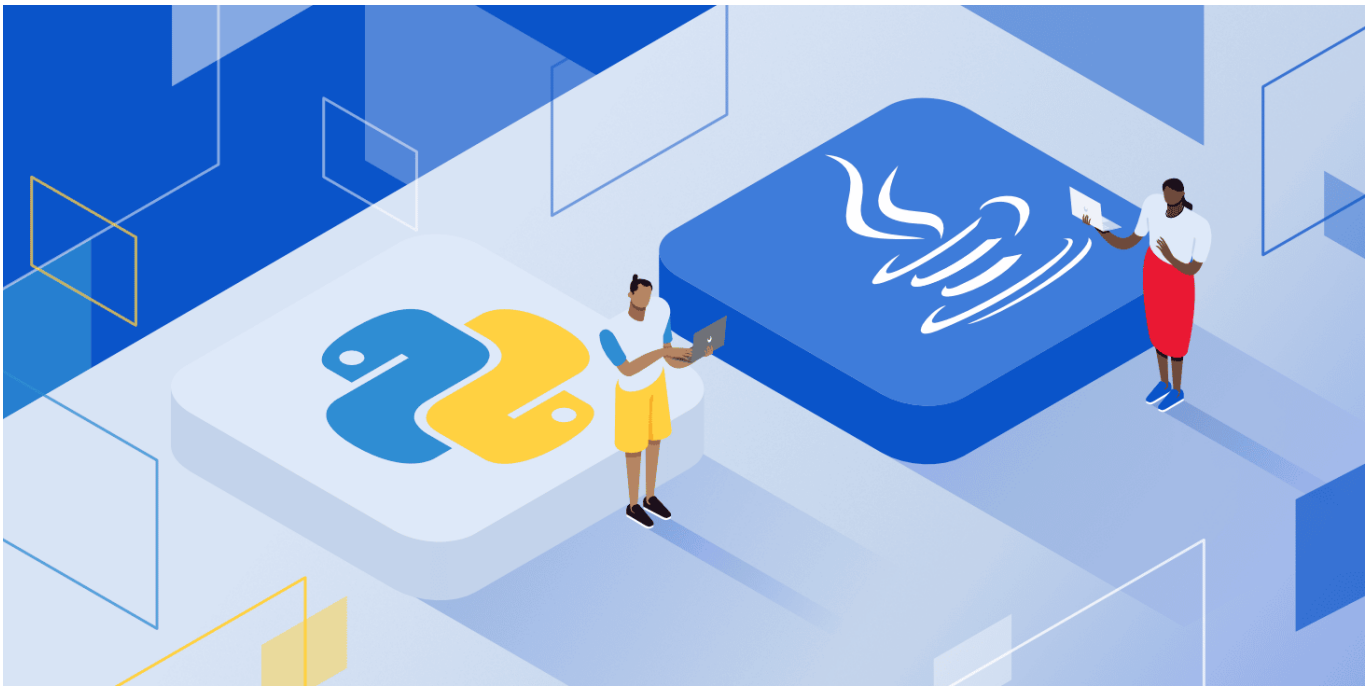

Bild von Kinsta

Verschiedene Datentypen in Java

Java unterstützt verschiedene Datentypen, die zum Speichern verschiedener Arten von Daten verwendet werden können. Die wichtigsten Datentypen in Java sind:

- `int` für ganze Zahlen
- `double` für Fließkommazahlen
- `boolean` für Wahrheitswerte
- `char` für einzelne Zeichen
- `String` für Zeichenketten

Hier sind Beispiele für jeden dieser Datentypen:

int

Der Datentyp `int` wird verwendet, um ganze Zahlen zu speichern. Hier ist ein Beispiel, das zwei Zahlen addiert und das Ergebnis auf der Konsole ausgibt:

```
int num1 = 5;
int num2 = 10;
int sum = num1 + num2;
System.out.println("Die Summe ist " + sum);
```

double

Der Datentyp `double` wird verwendet, um Fließkommazahlen zu speichern. Hier ist ein Beispiel, das den Durchschnitt von drei Zahlen berechnet und das Ergebnis auf der Konsole ausgibt:

```
double num1 = 5.5;
double num2 = 10.75;
double num3 = 3.25;
double average = (num1 + num2 + num3) / 3;
System.out.println("Der Durchschnitt ist " + average);
```

boolean

Der Datentyp `boolean` wird verwendet, um Wahrheitswerte zu speichern. Hier ist ein Beispiel, das prüft, ob eine Zahl gerade ist, und das Ergebnis auf der Konsole ausgibt:

```
int num = 4;
boolean isEven = (num % 2 == 0);
System.out.println("Ist die Zahl gerade? " + isEven);
```

char

Der Datentyp `char` wird verwendet, um einzelne Zeichen zu speichern. Hier ist ein Beispiel, das das erste Zeichen einer Zeichenkette extrahiert und auf der Konsole ausgibt:

```
String text = "Hello";
char firstChar = text.charAt(0);
System.out.println("Das erste Zeichen ist " + firstChar);
```

String

Der Datentyp `String` wird verwendet, um Zeichenketten zu speichern. Hier ist ein Beispiel, das zwei Zeichenketten verbindet und das Ergebnis auf der Konsole ausgibt:

```
String str1 = "Hello";
String str2 = "World";
String result = str1 + " " + str2;
System.out.println(result);
```

Wertzuweisungen / Mathematische Operatoren

Gerne! Hier sind einige Beispiele für Wertzuweisung und mathematische Operationen in Java:

Wertzuweisung

Die Wertzuweisung wird verwendet, um Werte einer Variablen zuzuweisen. In Java erfolgt die Wertzuweisung mit dem Gleichheitszeichen "=".

```
int num = 10; // Wert 10 wird der Variable "num" zugewiesen
String name = "Max"; // Der Wert "Max" wird der Variable "name" zugewiesen
boolean isTrue = false; // Der Wert "false" wird der Variable "isTrue" zugewiesen
```

Mathematische Operationen

Java unterstützt verschiedene mathematische Operationen wie Addition, Subtraktion, Multiplikation, Division und Modulo.

```
int num1 = 10;
int num2 = 5;
int sum = num1 + num2; // Addition
int difference = num1 - num2; // Subtraktion
int product = num1 * num2; // Multiplikation
int quotient = num1 / num2; // Division
int remainder = num1 % num2; // Modulo (Restwert)
```

Man kann auch Klammern verwenden, um die Reihenfolge der mathematischen Operationen zu steuern:

```
int result = ((num1 + num2) * 2) / num2; // Die Klammern stellen sicher, dass die
Addition zuerst ausgeführt wird
```

Das "+" -Zeichen kann auch verwendet werden, um Zeichenketten zu verbinden (konkatenieren):

```
String firstName = "Max";
String lastName = "Mustermann";
String fullName = firstName + " " + lastName; // Konkatenieren von zwei
Zeichenketten
```

Vergleichsoperatoren: ==, !=, >=, <=, <, >

== und !=

Der Vergleichsoperator "==" wird verwendet, um zu prüfen, ob zwei Werte gleich sind. Der Operator "!=" wird verwendet, um zu prüfen, ob zwei Werte ungleich sind.

```
int num1 = 5;
int num2 = 10;
```

```
boolean isEqual = (num1 == num2); // isEqual ist false, da num1 nicht gleich num2
ist
boolean isNotEqual = (num1 != num2); // isNotEqual ist true, da num1 ungleich num2
ist
```

>=, <=, < und >

Die Vergleichsoperatoren ">=", "<=", "<" und ">" werden verwendet, um zu prüfen, ob ein Wert größer oder kleiner als ein anderer Wert ist.

```
int num1 = 5;
int num2 = 10;
boolean isGreaterOrEqual = (num1 >= num2); // isGreaterOrEqual ist false, da num1
nicht größer oder gleich num2 ist
boolean isLessOrEqual = (num1 <= num2); // isLessOrEqual ist true, da num1 kleiner
oder gleich num2 ist
boolean isLess = (num1 < num2); // isLess ist true, da num1 kleiner als num2 ist
boolean isGreater = (num1 > num2); // isGreater ist false, da num1 nicht größer
als num2 ist
```

Man kann auch Vergleichsoperatoren mit booleschen Ausdrücken verwenden:

```
boolean isTrue = true;
boolean isFalse = false;
boolean isGreaterThanFalse = (isTrue > isFalse); // isGreaterThanFalse ist true,
da true größer als false ist (wahr ist größer als falsch)
```

Verknüpfungsoperatoren &&(AND), ||(OR)

&& (And)

Der Verknüpfungsoperator "&&" wird als "And" bezeichnet und wird verwendet, um zu prüfen, ob zwei Bedingungen gleichzeitig erfüllt sind.

```
int num1 = 5;
int num2 = 10;
boolean isGreaterThanZero = (num1 > 0);
boolean isLessThanTen = (num2 < 10);
boolean isBothTrue = (isGreaterThanZero && isLessThanTen); // isBothTrue ist
false, da isLessThanTen nicht erfüllt ist
```

|| (OR)

Der Verknüpfungsoperator "||" wird als "Or" bezeichnet und wird verwendet, um zu prüfen, ob mindestens eine Bedingung erfüllt ist.

```
int num1 = 5;
int num2 = 10;
boolean isGreaterThanZero = (num1 > 0);
boolean isLessThanTen = (num2 < 10);
boolean isEitherTrue = (isGreaterThanZero || isLessThanTen); // isEitherTrue ist
true, da isGreaterThanZero erfüllt ist
```

Man kann Verknüpfungsoperatoren auch mit Vergleichsoperatoren kombinieren:

```
int num1 = 5;
int num2 = 10;
int num3 = 15;
boolean isGreaterThanZero = (num1 > 0);
boolean isLessThanTen = (num2 < 10);
boolean isBetweenFiveAndFifteen = (num1 <= num3 && num3 <= num2); //
isBetweenFiveAndFifteen ist true, da num3 zwischen num1 und num2 liegt
```

if-else

```
int num = 10;

if (num > 0) { // Wenn die Bedingung erfüllt ist (true), wird dieser Block
ausgeführt
    System.out.println("Die Zahl ist positiv.");
} else { // Wenn die Bedingung nicht erfüllt ist (false), wird dieser Block
ausgeführt
    System.out.println("Die Zahl ist negativ oder gleich Null.");
}
```

In diesem Beispiel wird geprüft, ob die Variable "num" größer als Null ist. Wenn die Bedingung erfüllt ist, wird die erste Anweisung "Die Zahl ist positiv" auf der Konsole ausgegeben. Andernfalls wird die zweite Anweisung "Die Zahl ist negativ oder gleich Null" ausgegeben.

Man kann auch eine if-else-Anweisung mit mehreren Bedingungen verwenden:

```
int num = 10;

if (num > 0) {
    System.out.println("Die Zahl ist positiv.");
} else if (num < 0) {
```

```
    System.out.println("Die Zahl ist negativ.");
} else {
    System.out.println("Die Zahl ist Null.");
}
```

In diesem Beispiel wird geprüft, ob die Variable "num" größer als Null, kleiner als Null oder gleich Null ist. Je nachdem, welche Bedingung erfüllt ist, wird die entsprechende Anweisung ausgeführt.

switch-case

```
int num = 2;

switch (num) { // Die Variable, die geprüft werden soll
    case 1:
        System.out.println("Die Zahl ist eins.");
        break;
    case 2:
        System.out.println("Die Zahl ist zwei.");
        break;
    case 3:
        System.out.println("Die Zahl ist drei.");
        break;
    default:
        System.out.println("Die Zahl ist nicht eins, zwei oder drei.");
        break;
}
```

In diesem Beispiel wird geprüft, ob die Variable "num" den Wert 1, 2, 3 oder einen anderen Wert hat. Je nachdem, welcher Fall zutrifft, wird der entsprechende Block ausgeführt. Wenn keiner der Fälle zutrifft, wird der Standard-Block ausgeführt.

Es ist wichtig, das `break`-Statement am Ende jedes Blocks zu verwenden, um sicherzustellen, dass die Switch-Case-Anweisung verlassen wird, sobald der korrekte Fall gefunden wurde.

Man kann auch einen Fall für mehrere Werte erstellen:

```
int num = 2;

switch (num) {
    case 1:
    case 2:
    case 3:
        System.out.println("Die Zahl ist eins, zwei oder drei.");
        break;
    default:
        System.out.println("Die Zahl ist nicht eins, zwei oder drei.");
```

```
        break;
}
```

In diesem Beispiel wird geprüft, ob die Variable "num" den Wert 1, 2 oder 3 hat. Wenn ja, wird die erste Anweisung ausgeführt. Wenn nicht, wird der Standard-Block ausgeführt.

while()

```
int count = 1;

while (count <= 5) { // Die Schleife wird ausgeführt, solange die Bedingung
erfüllt ist
    System.out.println("Die Schleife wird zum " + count + ". Mal durchlaufen.");
    count++; // Die Schleifenvariable wird inkrementiert
}
```

In diesem Beispiel wird eine while-Schleife verwendet, um den Satz "Die Schleife wird zum x. Mal durchlaufen" fünfmal auf der Konsole auszugeben. Die Schleife wird solange ausgeführt, wie die Bedingung "count <= 5" erfüllt ist. Die Schleifenvariable "count" wird in jedem Schleifendurchlauf um eins inkrementiert.

Man kann auch eine Bedingung verwenden, um die Schleife vorzeitig zu beenden:

```
int count = 1;

while (count <= 10) {
    System.out.println("Die Schleife wird zum " + count + ". Mal durchlaufen.");
    if (count == 5) { // Wenn die Bedingung erfüllt ist, wird die Schleife beendet
        break;
    }
    count++;
}
```

In diesem Beispiel wird eine while-Schleife verwendet, um den Satz "Die Schleife wird zum x. Mal durchlaufen" auszugeben. Die Schleife wird jedoch vorzeitig beendet, wenn die Schleifenvariable "count" den Wert 5 erreicht. Dazu wird die `break`-Anweisung verwendet, die die Schleife verlässt, sobald die Bedingung erfüllt ist.

do .. while()

```
int count = 1;

do { // Der Schleifenkörper wird mindestens einmal ausgeführt
    System.out.println("Die Schleife wird zum " + count + ". Mal durchlaufen.");
    count++; // Die Schleifenvariable wird inkrementiert
} while (count <= 5); // Die Schleife wird ausgeführt, solange die Bedingung
erfüllt ist
```

In diesem Beispiel wird eine do-while-Schleife verwendet, um den Satz "Die Schleife wird zum x. Mal durchlaufen" fünfmal auf der Konsole auszugeben. Im Gegensatz zur while-Schleife wird der Schleifenkörper bei der do-while-Schleife mindestens einmal ausgeführt, bevor die Bedingung überprüft wird.

Man kann auch eine Bedingung verwenden, um die Schleife vorzeitig zu beenden:

```
int count = 1;

do {
    System.out.println("Die Schleife wird zum " + count + ". Mal durchlaufen.");
    if (count == 5) { // Wenn die Bedingung erfüllt ist, wird die Schleife beendet
        break;
    }
    count++; // Die Schleifenvariable wird inkrementiert
} while (count <= 10);
```

In diesem Beispiel wird eine do-while-Schleife verwendet, um den Satz "Die Schleife wird zum x. Mal durchlaufen" auszugeben. Die Schleife wird jedoch vorzeitig beendet, wenn die Schleifenvariable "count" den Wert 5 erreicht. Dazu wird die `break`-Anweisung verwendet, die die Schleife verlässt, sobald die Bedingung erfüllt ist.

for()

```
for (int i = 1; i <= 5; i++) { // Die Schleife wird fünfmal durchlaufen
    System.out.println("Die Schleife wird zum " + i + ". Mal durchlaufen.");
}
```

In diesem Beispiel wird eine for-Schleife verwendet, um den Satz "Die Schleife wird zum x. Mal durchlaufen" fünfmal auf der Konsole auszugeben. Die Schleife wird durch den Schleifenkopf gesteuert, der aus einer Initialisierung (`int i = 1`), einer Bedingung (`i <= 5`) und einer Inkrementierung (`i++`) besteht.

Man kann auch verschachtelte for-Schleifen verwenden, um mehrere Iterationen durchzuführen:

```
for (int i = 1; i <= 3; i++) { // Die äußere Schleife wird dreimal durchlaufen
    for (int j = 1; j <= 2; j++) { // Die innere Schleife wird zweimal durchlaufen
        System.out.println("Die äußere Schleife wird zum " + i + ". Mal und die
innere Schleife zum " + j + ". Mal durchlaufen.");
    }
}
```

In diesem Beispiel wird eine verschachtelte for-Schleife verwendet, um den Satz "Die äußere Schleife wird zum x. Mal und die innere Schleife zum y. Mal durchlaufen" sechsmal auf der Konsole auszugeben. Die äußere Schleife wird dreimal durchlaufen und die innere Schleife wird jedes Mal zweimal durchlaufen.

Math. Java library

```
double x = 2.0;
double y = 3.0;
double result1 = Math.pow(x, y); // x hoch y
double result2 = Math.sin(x); // Sinus von x

System.out.println("Das Ergebnis von " + x + " hoch " + y + " ist " + result1);
System.out.println("Der Sinus von " + x + " ist " + result2);
```

In diesem Beispiel werden die Funktionen `Math.pow()` und `Math.sin()` verwendet, um das Ergebnis von 2 hoch 3 und den Sinus von 2 zu berechnen. Die `Math`-Library bietet eine Vielzahl von mathematischen Funktionen, die in Java verwendet werden können.

Es ist wichtig, die richtige Schreibweise und Parameteranzahl für jede Funktion zu verwenden. Die `Math.pow()`-Funktion erwartet beispielsweise zwei Parameter: die Basis und den Exponenten, während die `Math.sin()`-Funktion nur einen Parameter erwartet: den Winkel in Radian.

Hier sind einige weitere Beispiele für Funktionen aus der `Math`-Library:

```
double x = 2.0;
double y = 3.0;

double result1 = Math.sqrt(x); // Quadratwurzel von x
double result2 = Math.log(x); // Natürlicher Logarithmus von x
double result3 = Math.max(x, y); // Größere von x und y
double result4 = Math.min(x, y); // Kleinere von x und y

System.out.println("Die Quadratwurzel von " + x + " ist " + result1);
System.out.println("Der natürliche Logarithmus von " + x + " ist " + result2);
System.out.println("Die größere Zahl von " + x + " und " + y + " ist " + result3);
System.out.println("Die kleinere Zahl von " + x + " und " + y + " ist " +
result4);
```

Arrays one-dimensional, two-dimensional

Gerne! Hier sind Beispiele für die Verwendung von for-Schleifen für Arrays in Java:

Eindimensionale Arrays

```
int[] numbers = {1, 2, 3, 4, 5};

// Iteration über ein eindimensionales Array mit einer for-Schleife
for (int i = 0; i < numbers.length; i++) {
```

```
        System.out.println("Element an Index " + i + ": " + numbers[i]);
    }
```

In diesem Beispiel wird ein eindimensionales Array "numbers" mit den Werten 1, 2, 3, 4 und 5 deklariert und initialisiert. Eine for-Schleife wird verwendet, um über das Array zu iterieren und jedes Element auf der Konsole auszugeben. Die Schleife wird durch den Schleifenkopf gesteuert, der von der Schleifenvariable `i` und der Länge des Arrays `numbers.length` abhängt.

Zweidimensionale Arrays

```
int[][] matrix = {{1, 2}, {3, 4}, {5, 6}};

// Iteration über ein zweidimensionales Array mit zwei verschachtelten for-
Schleifen
for (int i = 0; i < matrix.length; i++) {
    for (int j = 0; j < matrix[i].length; j++) {
        System.out.println("Element an Index (" + i + ", " + j + "): " + matrix[i]
[j]);
    }
}
```

In diesem Beispiel wird ein zweidimensionales Array "matrix" mit den Werten {{1, 2}, {3, 4}, {5, 6}} deklariert und initialisiert. Zwei verschachtelte for-Schleifen werden verwendet, um über das Array zu iterieren und jedes Element auf der Konsole auszugeben. Die äußere Schleife iteriert über die Zeilen des Arrays, während die innere Schleife über die Spalten iteriert. Jedes Element wird durch die Indizes `i` und `j` angegeben.

Maximum eines eindimensionalen Arrays finden

```
int[] numbers = {1, 5, 2, 7, 3};
int max = numbers[0];

for (int i = 1; i < numbers.length; i++) {
    if (numbers[i] > max) {
        max = numbers[i];
    }
}

System.out.println("Das Maximum ist: " + max);
```

In diesem Beispiel wird ein eindimensionales Array "numbers" mit den Werten 1, 5, 2, 7 und 3 deklariert und initialisiert. Eine for-Schleife wird verwendet, um das Maximum des Arrays zu finden. Die Variable "max" wird zunächst auf das erste Element des Arrays gesetzt. In jeder Schleifeniteration wird geprüft, ob das aktuelle Element größer als das bisherige Maximum ist. Wenn ja, wird "max" auf das aktuelle Element gesetzt. Am Ende der Schleife wird das Maximum auf der Konsole ausgegeben.

Arrayszugriff

Eindimensionale Arrays

```
int[] numbers = {1, 2, 3, 4, 5};

// Zugriff auf ein eindimensionales Array ohne Methoden oder Funktionen
for (int i = 0; i < numbers.length; i++) {
    System.out.println("Element an Index " + i + ": " + numbers[i]);
}
```

In diesem Beispiel wird ein eindimensionales Array "numbers" mit den Werten 1, 2, 3, 4 und 5 deklariert und initialisiert. Eine for-Schleife wird verwendet, um über das Array zu iterieren und jedes Element auf der Konsole auszugeben. Der Zugriff auf das Array erfolgt durch den Index `i` und den Array-Namen `numbers`.

Zweidimensionale Arrays

```
int[][] matrix = {{1, 2}, {3, 4}, {5, 6}};

// Zugriff auf ein zweidimensionales Array ohne Methoden oder Funktionen
for (int i = 0; i < matrix.length; i++) {
    for (int j = 0; j < matrix[i].length; j++) {
        System.out.println("Element an Index (" + i + ", " + j + "): " + matrix[i]
[j]);
    }
}
```

In diesem Beispiel wird ein zweidimensionales Array "matrix" mit den Werten {{1, 2}, {3, 4}, {5, 6}} deklariert und initialisiert. Zwei verschachtelte for-Schleifen werden verwendet, um über das Array zu iterieren und jedes Element auf der Konsole auszugeben. Der Zugriff auf ein Element des Arrays erfolgt durch die Indizes `i` und `j` und den Array-Namen `matrix`.

Maximum eines eindimensionalen Arrays finden

```
int[] numbers = {1, 5, 2, 7, 3};
int max = numbers[0];

// Finden des Maximums eines eindimensionalen Arrays ohne Methoden oder Funktionen
for (int i = 1; i < numbers.length; i++) {
    if (numbers[i] > max) {
        max = numbers[i];
    }
}

System.out.println("Das Maximum ist: " + max);
```

In diesem Beispiel wird ein eindimensionales Array "numbers" mit den Werten 1, 5, 2, 7 und 3 deklariert und initialisiert. Eine for-Schleife wird verwendet, um das Maximum des Arrays zu finden. Die Variable "max" wird zunächst auf das erste Element des Arrays gesetzt. In jeder Schleifeniteration wird geprüft, ob das aktuelle Element größer als das bisherige Maximum ist. Wenn ja, wird "max" auf das aktuelle Element gesetzt. Am Ende der Schleife wird das Maximum auf der Konsole ausgegeben.

Kommentare

```
// Einzeiliger Kommentar - dieser Kommentar wird von der Compiler-Software
ignoriert

/*
Mehrzeiliger Kommentar
Dieser Kommentar erstreckt sich über mehrere Zeilen
und wird ebenfalls von der Compiler-Software ignoriert
*/

/**
* Javadoc-Kommentar
* Dieser Kommentar wird für die Dokumentation von Klassen,
* Methoden und Feldern verwendet und kann vom Javadoc-Tool
* verarbeitet werden, um eine API-Dokumentation zu generieren.
*/

// Beispiel für die Verwendung von Kommentaren
public class Example {

    /**
    * Diese Methode gibt die Summe von zwei Zahlen zurück.
    * @param a die erste Zahl
    * @param b die zweite Zahl
    * @return die Summe von a und b
    */
    public static int sum(int a, int b) {
        // Hier wird ein Einzeiler-Kommentar verwendet, um den Zweck der folgenden
Zeile zu erklären
        int result = a + b;
        return result;
    }
}
```

In diesem Beispiel werden drei Arten von Kommentaren gezeigt: einzeilige Kommentare, mehrzeilige Kommentare und Javadoc-Kommentare. Einzeilige und mehrzeilige Kommentare dienen dazu, den Code für andere Entwickler zu kommentieren und zu erklären. Javadoc-Kommentare sind spezielle Kommentare, die für die Erstellung von API-Dokumentationen verwendet werden können.

Es ist wichtig, Kommentare sinnvoll und präzise zu verwenden, um den Code für andere Entwickler zu erleichtern und die Wartbarkeit zu verbessern. Ein gut kommentierter Code erleichtert es auch, Fehler zu

finden und zu beheben.
