Java Programmierung Basis:

- Lexikalische Elemente:
  - Java-Programme bestehen aus Unicode-Zeichen, die eine Vielzahl von Symbolen ermöglichen.
  - Kommentare in Java können einzeilig, mehrzeilig oder Dokumentationskommentare sein, jede mit spezifischer Syntax.
- Bezeichner:
  - Bezeichner sind Namen für Variablen, Klassen oder Methoden, die bestimmten Namenskonventionen folgen.
- Variablendeklaration:
  - Die Syntax zur Deklaration von Variablen umfasst die Angabe des Typs und des Namens.

Kontrollstrukturen

- If-Anweisungen:
  - Die if-Anweisung bewertet eine Bedingung und führt einen Codeblock aus, wenn diese wahr ist.
  - Der else-Zweig bietet einen alternativen Ausführungspfad.

```
public class IfExample {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;

    // Beispiel einer if-Anweisung
        if (a < b) {
            System.out.println("a ist kleiner als b");
        }

    // Beispiel einer if-else-Anweisung
        if (a > b) {
            System.out.println("a ist größer als b");
        } else {
            System.out.println("a ist nicht größer als b");
        }

    // Beispiel einer if-else if-else-Anweisung
        if (a < b) {
            System.out.println("a ist kleiner als b");
        } else if (a == b) {
            System.out.println("a ist gleich b");
        } else {
            System.out.println("a ist größer als b");
        }
```

```
}
```

Switch-Cases:

- switch-Anweisung: Eine Mehrfachverzweigung, die Code basierend auf dem Wert einer Variablen ausführt.

```
int tag = 3;
switch (tag) {
  case 1:
    System.out.println("Montag");
    break;
  case 2:
    System.out.println("Dienstag");
    break;
  case 3:
    System.out.println("Mittwoch");
    break;
  default:
    System.out.println("Ungültiger Tag");
}
```

```
}
```

Datenrepräsentation

- Alle Datentypen in Java haben eine feste Größe, die in Bits dargestellt wird.
- Java unterstützt vier ganzzahlige Datentypen (byte, short, int, long), die alle vorzeichenbehaftet sind.
- Fließkommatypen umfassen float (einfache Genauigkeit) und double (doppelte Genauigkeit).

```
int alter = 25; // Ganzzahlige Variable
double gehalt = 3500.50; // Fließkommazahl
String name = "Max Mustermann"; // Zeichenkette
boolean istStudent = false; // Boolesche Variable
final double PI = 3.14159; // Konstante für den Wert von Pi
```

Primitiver Datentyp	Größe	Wertebereich
boolean	undefiniert	true/false
char	16 Bit	0 **bis** +65.535
byte	8 Bit	-128 **bis** +127
short	16 Bit	-32.768 **bis** +32.767
int	32 Bit	-2^31 **bis** +2^31 - 1
long	64 Bit	-2^63 **bis** +2^63 - 1
float	32 Bit	1,40239846E-45 **bis** 3,40282347E+38
double	64 Bit	4,94065645841246544E-324 **bis** 1,79769131486231570E+308

Variablen und Konstanten

- Variablen sind durch einen Namen, eine Adresse und einen Wert definiert und können während der Programmausführung geändert werden.
- Konstanten werden mit dem Schlüsselwort final deklariert, was ihre Werte unveränderlich macht.

Operatoren

- Java umfasst verschiedene Operatoren für Zuweisungen, logische Operationen, relationale Vergleiche und arithmetische Berechnungen.

```
public class OperatorExample {
    public static void main(String[] args) {
// Arithmetische Operatoren
        int a = 10;
        int b = 5;

        int summe = a + b; // Addition
        int differenz = a - b; // Subtraktion
        int produkt = a * b; // Multiplikation
        double quotient = (double) a / b; // Division
        int rest = a % b; // Modulo

        System.out.println("Addition: " + summe);
        System.out.println("Subtraktion: " + differenz);
        System.out.println("Multiplikation: " + produkt);
        System.out.println("Division: " + quotient);
        System.out.println("Modulo: " + rest);

```

```
19      // Relationale Operatoren
20              boolean istGleich = (a == b); // Gleichheit
21              boolean istUngleich = (a != b); // Ungleichheit
22              boolean istGroesser = (a > b); // Größer
23              boolean istKleiner = (a < b); // Kleiner
24              boolean istGroesserGleich = (a >= b); // Größer oder gleich
25              boolean istKleinerGleich = (a <= b); // Kleiner oder gleich
26
27              System.out.println("Ist a gleich b? " + istGleich);
28              System.out.println("Ist a ungleich b? " + istUngleich);
29              System.out.println("Ist a größer als b? " + istGroesser);
30              System.out.println("Ist a kleiner als b? " + istKleiner);
31              System.out.println("Ist a größer oder gleich b? " + istGroesserGleich);
32              System.out.println("Ist a kleiner oder gleich b? " + istKleinerGleich);
33
34      // Logische Operatoren
35              boolean c = true;
36              boolean d = false;
37
38              boolean und = c && d; // Logisches UND
39              boolean oder = c || d; // Logisches ODER
40              boolean nicht = !c; // Logisches NICHT
41
42              System.out.println("c UND d: " + und);
43              System.out.println("c ODER d: " + oder);
44              System.out.println("NICHT c: " + nicht);
45      }
```

- ```
  46}
  ```

Typumwandlung

- Typumwandlung ist die Konvertierung eines Datentyps in einen anderen, die entweder erweiternd (sicher) oder einschränkend (potenziell unsicher) sein kann.

```
public class TypeCastingExample {
2     public static void main(String[] args) {
3       // Erweiternde Typumwandlung (Widening)
4               int intValue = 100;
5               double doubleValue = intValue; // int wird automatisch in double
        umgewandelt
6
7               System.out.println("Ursprünglicher int-Wert: " + intValue);
8               System.out.println("Erweiterte double-Wert: " + doubleValue);
```
```

```
9
10      // Verengende Typumwandlung (Narrowing)
11          double anotherDoubleValue = 9.78;
12          int anotherIntValue = (int) anotherDoubleValue; // double muss explizit in
        int umgewandelt werden
13
14          System.out.println("Ursprünglicher double-Wert: " + anotherDoubleValue);
15          System.out.println("Verengte int-Wert: " + anotherIntValue); //
        Nachkommastellen werden abgeschnitten
16
17      // Typumwandlung zwischen char und int
18          char charValue = 'A';
19          int charToInt = (int) charValue; // char wird in int umgewandelt
20
21          System.out.println("Ursprünglicher char-Wert: " + charValue);
22          System.out.println("char in int umgewandelter Wert: " + charToInt); // ASCII-
        Wert von 'A' ist 65
23
24      // Typumwandlung zwischen int und char
25          int intFromChar = 66; // ASCII-Wert für 'B'
26          char intToChar = (char) intFromChar; // int wird in char umgewandelt
27
28          System.out.println("Ursprünglicher int-Wert: " + intFromChar);
29          System.out.println("int in char umgewandelter Wert: " + intToChar); // 'B'
30      }
```

```
31}
```

Schleifen in Java

- `while-Schleife:`
  - Vorab-Test-Schleife, die so lange fortgesetzt wird, wie eine Bedingung wahr ist.
- do-while-Schleife:
  - Nach-Test-Schleife, die mindestens einmal ausgeführt wird, bevor die Bedingung überprüft wird.
- for-Schleife:
  - Kombiniert Initialisierung, Bedingungsprüfung und Aktualisierung in einer Zeile.
  - Ermöglicht eine prägnante Iteration über einen Wertebereich.
- Kontrollflussanweisungen:
- `break: Beendet die Schleife sofort.`
- `continue`: Überspringt die aktuelle Iteration und fährt mit der nächsten fort.

While-Schleife

```
int i = 0;
2
3while (i < 5) {
```

```
4       System.out.println("i ist: " + i);
5       i++; // Erhöht i um 1
6}
```

Do-while-Schleife:

```
int j = 0;
2
3do {
4       System.out.println("j ist: " + j);
5       j++; // Erhöht j um 1
6} while (j < 5);
```

For-Schleife:

```
for (int k = 0; k < 5; k++) {
2       System.out.println("k ist: " + k);
3}
```

Verschachtelte Schleifen:

```
for (int m = 0; m < 3; m++) {
2     for (int n = 0; n < 2; n++) {
3           System.out.println("m ist: " + m + ", n ist: " + n);
4     }
5}
```

5. Verwendung von break und continue

Die break-Anweisung wird verwendet, um eine Schleife vorzeitig zu beenden.

```
for (int p = 0; p < 10; p++) {
2     if (p == 5) {
3           break; // Beendet die Schleife, wenn p gleich 5 ist
4     }
5       System.out.println("p ist: " + p);
6}
```

Die continue-Anweisung wird verwendet, um die aktuelle Iteration zu überspringen und mit der nächsten fortzufahren.

```
for (int q = 0; q < 10; q++) {
2     if (q % 2 == 0) {
3           continue; // Überspringt die aktuelle Iteration, wenn q gerade ist
4     }
5       System.out.println("q ist: " + q);
6}
```

Datenstrukturen in Java:

- Arrays:
  - Java unterstützt eindimensionale und mehrdimensionale Arrays.
  - Arrays sind dynamisch zugewiesen, d.h. ihre Größe wird zur Laufzeit bestimmt.
- Syntax zur Deklaration von Arrays umfasst die Verwendung von eckigen Klammern (z.B. int[] a;).
  - Arrays können mit dem new-Operator oder durch literale Initialisierung initialisiert werden.

```
// Deklaration eines eindimensionalen Arrays
int[] zahlen = new int[5]; // Array mit 5 Elementen

// Initialisierung der Array-Elemente
zahlen[0] = 10;
zahlen[1] = 20;
zahlen[2] = 30;
zahlen[3] = 40;
zahlen[4] = 50;

// Ausgabe der Array-Elemente
for (int i = 0; i < zahlen.length; i++) {
    System.out.println("Element " + i + ": " + zahlen[i]);
}
```

```
// Deklaration und Initialisierung in einer Zeile
String[] farben = {"Rot", "Grün", "Blau"};

// Ausgabe der Array-Elemente
for (String farbe : farben) {
    System.out.println("Farbe: " + farbe);
}
```

- Zugriff auf Array-Elemente:
  - Elemente werden über ihren Index angesprochen, beginnend bei 0.
  - Die Länge eines Arrays kann mit der length-Eigenschaft abgerufen werden.

```
int[] zahlen = {10, 20, 30, 40, 50};

// Zugriff auf ein bestimmtes Element
int drittesElement = zahlen[2]; // Index 2 entspricht dem dritten Element
System.out.println("Das dritte Element ist: " + drittesElement);
```

Mehrdimensionale Arrays:

- Werden mit mehreren Paaren von eckigen Klammern deklariert (z.B. int[][] a;).

- Können nicht-rechteckige Datenstrukturen darstellen.

```
// Deklaration eines zweidimensionalen Arrays
int[][] matrix = new int[2][3]; // 2 Zeilen und 3 Spalten

// Initialisierung der Array-Elemente
matrix[0][0] = 1;
matrix[0][1] = 2;
matrix[0][2] = 3;
matrix[1][0] = 4;
matrix[1][1] = 5;
matrix[1][2] = 6;

// Ausgabe der Array-Elemente
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(); // Neue Zeile nach jeder Zeile der Matrix
}
```

- Schlüsselkonzepte:
  - Klasse: Eine Vorlage zur Erstellung von Objekten mit gemeinsamen Eigenschaften.
  - Objekt: Eine konkrete Instanz einer Klasse.
  - Attribut: Daten, die den Zustand eines Objekts beschreiben.
  - Operation: Eine Funktion, die das Verhalten eines Objekts definiert.
- Beziehung zwischen Klasse und Objekt:
  - Klassen dienen als Vorlagen zur Erstellung von Objekten, die ihre eigenen Attribute und Methoden haben.
  - Beispiel: Eine Klasse "Auto" kann Attribute wie Name, Jahr der Erstzulassung und Leistung haben.

- Syntax für Klassendefinition:
  - Allgemeine Struktur: [Modifikator] class Klassenname [extends Basisklasse] [implements Schnittstellen] { Attributdeklarationen; Operationsdeklarationen; }
  - Beispiel einer einfachen Klasse:

java

```
class Auto {
    String name;
    int jahrDerErstzulassung;
    int leistung;
```

```
5}
```

- Objekterstellung:
  - Objekte werden mit dem new-Operator instanziiert.
  - Beispiel der Objekterstellung:

```
java
□□
1Auto meinPkw = new Auto();
```