Methoden in Java: Grundlagen, Aufbau, Aufruf, Parameter, Argumente und Wertübergabe

In Java sind **Methoden** wiederverwendbare Codeblöcke, die eine bestimmte Aufgabe oder Berechnung ausführen. Sie bestehen aus einem definierten Namen, einem Rückgabetyp, optionalen Parametern und einem Methodenrumpf, der die Anweisungen enthält.

Grundlagen der Methoden

Eine **Methode** in Java besteht aus vier Hauptbestandteilen:

1. **Rückgabetyp**: Der Datentyp, den die Methode zurückgibt, oder `void`, wenn keine Rückgabe erfolgt.
2. **Methodenname**: Der Name der Methode, der ihren Zweck beschreibt und durch den sie aufgerufen wird.
3. **Parameter**: Optional; Eingabewerte, die der Methode übergeben werden.
4. **Methodenrumpf**: Der Codeblock, der die Anweisungen der Methode enthält.

Syntax für eine Methode

```
Rückgabetyp methodenName(Parameterliste) {
    // Methodenrumpf
}
```

Einfache Methoden

Beispiel: Methode ohne Rückgabewert (`void`)

```
public class BeispielMethoden {
    // Methode ohne Rückgabewert, die einen Text ausgibt
    public void begruessung() {
        System.out.println("Hallo, willkommen zur Java-Programmierung!");
    }
}
```

Erklärung:

- `void` zeigt an, dass die Methode nichts zurückgibt.
- `begruessung` ist der Methodenname.
- Der Methodenrumpf enthält eine einfache Anweisung, die eine Begrüßung ausgibt.

Aufruf der Methode

```
```

```
public class Main {
    public static void main(String[] args) {
        BeispielMethoden obj = new BeispielMethoden(); // Erstellen eines Objekts
der Klasse
        obj.begruessung(); // Aufruf der Methode
    }
}
```

Methoden mit Rückgabewert

Beispiel: Methode mit Rückgabewert (`int`)

```
public class Rechner {
    // Methode mit Rückgabewert vom Typ int
    public int addiere(int zahl1, int zahl2) {
        int summe = zahl1 + zahl2;
        return summe; // Rückgabe der Summe
    }
}
```

Erklärung:

- `int` ist der Rückgabetyp, daher erwartet der Methodenaufruf eine `int`-Rückgabe.
- `addiere` ist der Methodenname.
- `int zahl1, int zahl2` sind Parameter, die bei jedem Methodenaufruf übergeben werden müssen.
- `return` gibt die Berechnungsergebnisse zurück.

Aufruf der Methode

```
public class Main {
    public static void main(String[] args) {
        Rechner rechner = new Rechner(); // Erstellen eines Objekts
        int ergebnis = rechner.addiere(5, 3); // Aufruf der Methode mit Argumenten
        System.out.println("Die Summe ist: " + ergebnis); // Ausgabe der Summe
    }
}
```

Überladung von Methoden (Method Overloading)

Java unterstützt **Methodenüberladung** (Overloading), d.h., es können mehrere Methoden mit demselben Namen, jedoch unterschiedlichen Parametertypen oder -anzahlen, erstellt werden.

Beispiel: Überladene Methoden

```java
public class Ueberladen {
    // Methode für zwei Parameter
    public int addiere(int zahl1, int zahl2) {
        return zahl1 + zahl2;
    }

    // Methode für drei Parameter
    public int addiere(int zahl1, int zahl2, int zahl3) {
        return zahl1 + zahl2 + zahl3;
    }
}
```

Erklärung:

- Beide Methoden heißen `addiere`, unterscheiden sich jedoch durch die Anzahl der Parameter.
- Der Compiler entscheidet, welche Methode aufzurufen ist, basierend auf den übergebenen Argumenten.

Aufruf der Methode

```java
public class Main {
    public static void main(String[] args) {
        Ueberladen rechner = new Ueberladen();
        int ergebnisZwei = rechner.addiere(5, 10); // Aufruf der Methode mit zwei Parametern
        int ergebnisDrei = rechner.addiere(5, 10, 15); // Aufruf der Methode mit drei Parametern

        System.out.println("Summe von zwei Zahlen: " + ergebnisZwei);
        System.out.println("Summe von drei Zahlen: " + ergebnisDrei);
    }
}
```

Parameter, Argumente und Wertübergabe

Parameter und Argumente

Parameter sind Platzhalter in der Methodendeklaration, die durch **Argumente** beim Aufruf der Methode ersetzt werden. Beispiel:

```java
public int multipliziere(int zahl1, int zahl2) { // zahl1 und zahl2 sind Parameter
    return zahl1 * zahl2;
}

// Aufruf der Methode mit Argumenten
int ergebnis = multipliziere(5, 3); // 5 und 3 sind Argumente
```

Wertübergabe

1. **Call by Value**: Für primitive Datentypen – es wird nur der Wert übergeben, Änderungen in der Methode wirken sich nicht auf die ursprüngliche Variable aus.

```
public void aendereWert(int zahl) {
    zahl = zahl + 10;
    System.out.println("Innerhalb der Methode: " + zahl); // Innerhalb der
Methode: 20
}

int original = 10;
aendereWert(original);
System.out.println("Außerhalb der Methode: " + original); // Außerhalb der
Methode: 10
```

Statische Methoden (`static`)

Definition: Statische Methoden gehören zur Klasse selbst und können ohne Erstellen einer Instanz (eines Objekts) der Klasse aufgerufen werden. Sie werden mit dem `static`-Schlüsselwort definiert.

Eigenschaften von Statischen Methoden

- **Zugehörigkeit zur Klasse**: Eine statische Methode gehört zur Klasse und nicht zu einer spezifischen Instanz (Objekt) der Klasse.
- **Zugriff über Klassennamen**: Statische Methoden werden häufig über den Klassennamen aufgerufen (`Klassenname.methodenName()`).
- **Zugriffsbeschränkungen**: Statische Methoden können **nur auf andere statische Variablen und Methoden** innerhalb derselben Klasse zugreifen.
- **Speicher**: Statische Methoden werden einmal im Speicher abgelegt und bleiben dort für die Laufzeit des Programms.

Beispiel für eine Statische Methode

```
public class MatheUtils {
    // Statische Methode, um das Quadrat einer Zahl zu berechnen
    public static int berechneQuadrat(int zahl) {
        return zahl * zahl;
    }
}

public class Main {
    public static void main(String[] args) {
        // Aufruf der statischen Methode ohne Erstellen einer Instanz von
MatheUtils
        int ergebnis = MatheUtils.berechneQuadrat(5);
```

```
        System.out.println("Das Quadrat ist: " + ergebnis); // Das Quadrat ist: 25
    }
}
```

Nicht-Statische Methoden (Instanzmethoden)

Definition: Nicht-statische Methoden (auch Instanzmethoden genannt) gehören zu einer Instanz (einem Objekt) der Klasse. Sie benötigen eine Instanz, um aufgerufen zu werden.

Eigenschaften von Nicht-Statischen Methoden

- **Zugehörigkeit zur Instanz**: Sie sind an eine spezifische Instanz der Klasse gebunden.
- **Zugriff auf Instanzvariablen und Methoden**: Nicht-statische Methoden können auf **alle Instanzvariablen und Instanzmethoden** der Klasse zugreifen.
- **Aufruf durch Instanziierung**: Eine Instanz der Klasse muss zuerst erstellt werden, bevor die Methode aufgerufen werden kann.

Beispiel für eine Nicht-Statische Methode

```
public class Kreis {
    // Instanzvariable für den Radius
    public double radius;

    // Nicht-statische Methode zur Berechnung des Kreisumfangs
    public double berechneUmfang() {
        return 2 * Math.PI * radius;
    }
}

public class Main {
    public static void main(String[] args) {
        // Erstellen einer Instanz der Klasse Kreis
        Kreis kreis1 = new Kreis();
        kreis1.radius = 5.0;  // Direkte Zuweisung des Radius

        // Aufruf der nicht-statischen Methode über die Instanz
        double umfang = kreis1.berechneUmfang();
        System.out.println("Der Umfang des Kreises ist: " + umfang); // Ausgabe
des Umfangs
    }
}
```

Zusammenfassung der Methoden in Java

Merkmal	Statische Methode	Nicht-Statische Methode
Zugehörigkeit	Zur Klasse	Zur Instanz (dem Objekt) der Klasse

Merkmal	Statische Methode	Nicht-Statische Methode
Aufruf	Über den Klassennamen	Über die Instanz der Klasse
Zugriff auf Instanzvariablen	Nein	Ja
Anwendung	Für allgemeine, unveränderliche Funktionen, Utility-Methoden	Methoden, die auf Objektdaten zugreifen und sie verändern

Java-Methoden bieten eine strukturierte und wiederverwendbare Möglichkeit, Code zu organisieren und zu modularisieren. Statische Methoden eignen sich für allgemeine, unveränderliche Aufgaben und nicht-statische Methoden sind nützlich, wenn sie auf Instanzdaten zugreifen und diese verändern sollen.