Sie sind nützlich, wenn du bereits eine Methode hast, deren Signatur dem erwarteten funktionalen Interface entspricht, und du diese Methode direkt als Lambda verwenden möchtest.

Warum Methoden-Referenzen?

Normalerweise schreiben wir Lambda-Ausdrücke, um eine anonyme Funktion bereitzustellen. Wenn jedoch bereits eine Methode existiert, die genau das tut, was der Lambda-Ausdruck tun soll, können Methoden-Referenzen den Code kürzer und lesbarer machen.

Syntax der Methoden-Referenzen

Es gibt drei Hauptarten von Methoden-Referenzen in Java:

1. **Statische Methoden-Referenzen** (`Klasse::Methode`)
2. **Nicht-statische Methoden-Referenzen** (`Referenz::Methode`)
3. **Konstruktor-Referenzen** (`Klasse::new`)

1. Statische Methoden-Referenzen

Eine **statische Methoden-Referenz** verweist auf eine statische Methode einer Klasse. Die Syntax lautet `Klasse::Methode`. Dies ist nützlich, wenn du eine statische Methode als Implementierung für ein funktionales Interface verwenden möchtest.

Beispiel

Angenommen, wir haben eine Liste von `String`-Werten und möchten jeden Wert in einen `Integer` konvertieren. Anstatt einen Lambda-Ausdruck zu schreiben, können wir die statische Methode `Integer.parseInt` verwenden.

```java
import java.util.Arrays;
import java.util.List;

public class MethodenReferenzBeispiel {
    public static void main(String[] args) {
        List<String> zahlenStrings = Arrays.asList("1", "2", "3", "4");

        // Statische Methoden-Referenz zu Integer::parseInt
        zahlenStrings.stream()
                     .map(Integer::parseInt) // Integer.parseInt wird für jedes Element au
                     .forEach(System.out::println);
    }
}
```

Hier entspricht `Integer::parseInt` der Methode `parseInt` in der Klasse `Integer`. Das Lambda `(String s) -> Integer.parseInt(s)` wird durch `Integer::parseInt` ersetzt.

2. Nicht-statische Methoden-Referenzen

Eine **nicht-statische Methoden-Referenz** bezieht sich auf eine Instanzmethode eines Objekts. Die Syntax lautet `Referenz::Methode`. Diese Methode wird auf einem spezifischen Objekt aufgerufen, wenn der Lambda-Ausdruck ausgeführt wird.

Beispiel

Angenommen, wir möchten eine Liste von Zeichenfolgen in Großbuchstaben umwandeln. Anstatt `s -> s.toUpperCase()` zu verwenden, können wir eine Methoden-Referenz auf `toUpperCase` verwenden.

```java
import java.util.Arrays;
import java.util.List;

public class MethodenReferenzBeispiel {
    public static void main(String[] args) {
        List<String> worte = Arrays.asList("hallo", "welt", "java");

        // Nicht-statische Methoden-Referenz
        worte.stream()
             .map(String::toUpperCase) // "hallo"::toUpperCase, "welt"::toUpperCase, etc.
             .forEach(System.out::println);
    }
}
```

Hier entspricht `String::toUpperCase` einem Lambda-Ausdruck, der jedes `String`-Objekt in Großbuchstaben umwandelt. Dies ist äquivalent zu `s -> s.toUpperCase()`.

Methoden-Referenz auf eine spezifische Instanz

Es ist auch möglich, eine Methoden-Referenz auf eine spezifische Instanz zu verwenden:

```java
import java.util.Arrays;
import java.util.List;

public class MethodenReferenzBeispiel {
    public static void main(String[] args) {
        List<String> worte = Arrays.asList("eins", "zwei", "drei");

        // Spezifische Instanz-Methoden-Referenz
        MethodenReferenzBeispiel beispiel = new MethodenReferenzBeispiel();
        worte.forEach(beispiel::druckeWort);
    }

    public void druckeWort(String wort) {
        System.out.println("Wort: " + wort);
    }
}
```

Hier bezieht sich `beispiel::druckeWort` auf eine Instanzmethode der Instanz `beispiel` von `MethodenReferenzBeispiel`.

3. Konstruktor-Referenzen

Mit **Konstruktor-Referenzen** können wir neue Objekte erstellen. Die Syntax lautet `Klasse::new`. Dies ist besonders nützlich, wenn ein funktionales Interface ein Objekt zurückgeben soll.

Beispiel

Angenommen, wir möchten eine Liste von `String`-Werten in eine Liste von `Person`-Objekten umwandeln. Wir können den `Person`-Konstruktor referenzieren, um dies zu tun.

```java
class Person {
    String name;

    Person(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }
}

public class MethodenReferenzBeispiel {
    public static void main(String[] args) {
        List<String> namen = Arrays.asList("Alice", "Bob", "Charlie");

        // Konstruktor-Referenz
        List<Person> personen = namen.stream()
                                     .map(Person::new) // Erzeugt für jeden Namen ein neue
                                     .collect(Collectors.toList());

        personen.forEach(System.out::println); // Ausgabe: Alice, Bob, Charlie
    }
}
```

In diesem Fall wird `Person::new` verwendet, um neue `Person`-Objekte zu erstellen. Dies entspricht dem Lambda-Ausdruck `name -> new Person(name)`.

Vorteile von Methoden-Referenzen

- **Kürzerer und lesbarer Code**: Methoden-Referenzen machen den Code oft kürzer und klarer.
- **Wiederverwendung vorhandener Methoden**: Du kannst bestehende Methoden direkt verwenden, anstatt ähnliche Logik in einem neuen Lambda auszudrücken.
- **Weniger Fehleranfälligkeit**: Methoden-Referenzen nutzen bereits getestete Methoden, was weniger Fehleranfälligkeit bedeutet.

Zusammengefasst

1. **Statische Methoden-Referenzen** (`Klasse::Methode`) verwenden eine statische Methode direkt.
2. **Nicht-statische Methoden-Referenzen** (`Referenz::Methode`) verwenden eine Instanzmethode auf einem bestimmten Objekt.
3. **Konstruktor-Referenzen** (`Klasse::new`) verwenden einen Konstruktor, um neue Objekte zu erzeugen.