Lernskript: Lambda-Ausdrücke & Funktionale Interfaces

Die Einführung von Lambda-Ausdrücken in Java 8 hat die Sprache grundlegend modernisiert und den Weg für funktionale Programmierung geebnet.

1. Das Problem (Klassisch, vor Java 8)

Vor Java 8 war der Code oft sehr "geschwätzig" (verbose). Um ein einfaches Verhalten an eine Methode zu übergeben, musste man eine umständliche **anonyme innere Klasse** erstellen.

```
// Klassisch - sehr umständlich
List<String> namen = new ArrayList<>(List.of("Peter", "Anna", "Chris"));
Collections.sort(namen, new Comparator<String>() { // Anonyme innere Klasse
    @Override
    public int compare(String s1, String s2) {
        return Integer.compare(s1.length(), s2.length());
    }
});


2. Die Lösung (Modern, mit Lambdas, seit Java 8)

Ein Lambda-Ausdruck ist eine kurze, anonyme Funktion, die direkt im Code definiert
und übergeben werden kann.
Generated java


// Modern - kurz und prägnant
List<String> namen = new ArrayList<>(List.of("Peter", "Anna", "Chris"));
Collections.sort(namen, (s1, s2) -> Integer.compare(s1.length(), s2.length()));



Anatomie eines Lambdas:

    (s1, s2): Die Parameterliste (Typen werden oft vom Compiler inferiert).

    ->: Der Pfeil-Operator, der Parameter vom Körper trennt.

    ...: Der Körper der Funktion. Bei einer einzelnen Anweisung können {} und
return entfallen.

3. Funktionale Interfaces

Lambdas benötigen einen "Ziel-Typ". Dieser muss ein Funktionales Interface sein.

    Definition: Ein Interface, das genau eine einzige abstrakte Methode
```

deklariert.

    Annotation: @FunctionalInterface wird empfohlen, damit der Compiler die Regel prüft.

    Wichtige Beispiele aus java.util.function:

        Predicate<T>: Methode boolean test(T t). Prüft eine Bedingung.

        Function<T, R>: Methode R apply(T t). Wandelt ein Objekt um.

        Consumer<T>: Methode void accept(T t). Führt eine Aktion mit einem Objekt aus.

        Supplier<T>: Methode T get(). Erzeugt/liefert ein Objekt.

        Comparator<T>: Methode int compare(T o1, T o2). Vergleicht zwei Objekte.

4. Vertiefung (JavaMasta's Profi-Tipps)
Methodenreferenzen

Wenn ein Lambda nur eine einzige, bereits existierende Methode aufruft, kann man die noch kürzere Methodenreferenz-Syntax (::) verwenden. Das ist extrem lesbar und elegant.
Generated java

```
// Lambda-Version
namen.forEach(s -> System.out.println(s));

// Methodenreferenz-Version
namen.forEach(System.out::println);
```

Effektive Finalität (Effectively Final)

Eine lokale Variable, auf die ein Lambda von außerhalb zugreift, muss entweder final sein oder darf nach ihrer ersten Zuweisung nicht mehr verändert werden ("effektiv final"). Dies verhindert unerwartetes Verhalten und sorgt für Thread-Sicherheit.
Lambdas sind keine Magie

Der Compiler wandelt den Lambda-Ausdruck hinter den Kulissen in ein Objekt um, das das funktionale Interface implementiert. Für die JVM ist das Ergebnis ähnlich wie bei der alten anonymen Klasse, aber für den Entwickler ist die Syntax unendlich viel angenehmer und prägnanter.