Lernskript: Das Schlüsselwort `final`

Teil 1: Die grundlegende Analogie - Das "Warum"

- **Konzept:** `final` ist Javas Art, ein Versprechen abzugeben: **"Dieses Ding kann nach seiner ersten Zuweisung nie wieder verändert werden."** Der Compiler erzwingt dieses Versprechen.
- **Analogie:** Das **Geburtsdatum** auf einer Geburtsurkunde ist `final`. Der **Wohnort** ist es nicht.
- `final` kann an drei Stellen verwendet werden: bei **Variablen**, **Methoden** und **Klassen**.

Teil 2: Praktische Anwendungsfälle - Das "Wofür"

Use Case 1: `final`-Variablen (Konstanten & Unveränderlichkeit)

- **`static final`:** Definiert eine globale, unveränderliche **Konstante** für eine Klasse (z.B. `public static final double MEHRWERTSTEUER_SATZ = 0.19;`).
- **`final` Instanzfeld:** Stellt sicher, dass eine Eigenschaft eines Objekts (z.B. `private final int id;`) nach der Erstellung im Konstruktor **niemals mehr geändert** werden kann.
- **Wichtiger Sonderfall `final`-Referenzen:** Eine `final`-Referenzvariable (z.B. `final List<String> liste`) muss immer auf **dasselbe Objekt** zeigen. Der **interne Zustand des Objekts** (z.B. durch `liste.add("A")`) kann aber weiterhin verändert werden.

Use Case 2: `final`-Methoden (Überschreiben verhindern)

- **Problem:** Eine kritische Methode (z.B. `validiere()`) soll in Subklassen nicht überschrieben (`@Override`) werden können, um die Kernlogik zu schützen.
- **Lösung:** Die Methode als `final` deklarieren (`public final boolean validiere(...)`).

Use Case 3: `final`-Klassen (Vererbung komplett verbieten)

- **Problem:** Eine Klasse (wie `String`) ist in ihrem Design abgeschlossen und soll nicht erweitert werden können.
- **Lösung:** Die gesamte Klasse als `final` deklarieren (`public final class String`).

Teil 3: Wichtige Regeln & Vertiefung (Profi-Tipps)

- **Design > Performance:** Der Hauptgrund für `final` ist heute **sicheres und klares Softwaredesign**, nicht Mikro-Optimierung der Performance. Es kommuniziert die Absicht des Entwicklers.
- **`final`-Parameter:** `public void methode(final String name)` verhindert, dass der Parameter `name` innerhalb der Methode neu zugewiesen wird.
- **Unveränderliche Objekte (Immutability):** Klassen, deren Felder alle `private` und `final` sind und die keine Setter haben, sind **unveränderlich (immutable)**. Sie sind von Natur aus Thread-sicher und leichter zu verstehen. Moderne `Records` sind ein gutes Beispiel dafür.