Lernskript: Die `Object`-Klasse (`equals`, `hashCode`, `toString`)

Teil 1: Die grundlegende Analogie - Das "Warum"

- **Konzept:** JEDE Klasse in Java erbt unsichtbar von der "Urmutter" aller Klassen: `java.lang.Object`. Dadurch erbt jedes Objekt grundlegende Fähigkeiten, die wir oft anpassen (überschreiben) müssen, um sie nützlich zu machen.
- **Die 3 wichtigsten geerbten Fähigkeiten:**
    1. `equals(Object obj)` **(Der Vergleich):** Standardmäßig prüft diese Methode nur die Referenz-Gleichheit (`==`), also ob es sich um exakt dasselbe Objekt im Speicher handelt.
    2. `hashCode()` **(Der Aktenschrank-Code):** Liefert eine Integer-Zahl, die von Datenstrukturen wie `HashSet` und `HashMap` benutzt wird, um Objekte schnell in "Schubladen" zu sortieren und wiederzufinden.
    3. `toString()` **(Die Selbstbeschreibung):** Liefert standardmäßig eine kryptische Zeichenkette aus Klassenname und Speicheradresse (z.B. `Benutzer@1a2b3c4d`).
- **Der `equals`-`hashCode`-Vertrag:** Dies ist ein unumstößliches Gesetz in Java: **Wenn zwei Objekte laut `equals()` gleich sind, MÜSSEN sie exakt denselben `hashCode()`-Wert zurückgeben.** Ein Bruch dieses Vertrags führt zu unvorhersehbarem Verhalten in `HashSet` und `HashMap`.

Teil 2: Praktische Anwendungsfälle - Das "Wofür"

Use Case 1: Objekte in einem `HashSet` verwenden

- **Problem:** Ein `HashSet` soll Duplikate verhindern. Ohne überschriebenes `equals`/`hashCode` erkennt es zwei inhaltlich gleiche Objekte als unterschiedlich, weil sie unterschiedliche Speicheradressen haben.
- **Lösung:** `equals()` überschreiben, um einen inhaltlichen Vergleich (z.B. anhand einer ID oder ISBN) durchzuführen. `hashCode()` überschreiben, sodass es auf denselben Feldern wie `equals()` basiert.

Use Case 2: Sinnvolle Debug- und Log-Ausgaben

- **Problem:** `System.out.println(meinObjekt)` liefert eine nutzlose Ausgabe.
- **Lösung:** `toString()` überschreiben, um eine aussagekräftige, textuelle Repräsentation des Objektzustands zu liefern (z.B. `Benutzer{name='Anna', alter=30}`).

Use Case 3: Die moderne Abkürzung

- **Problem:** Das manuelle Schreiben dieser Methoden ist mühsam und fehleranfällig.
- **Lösung A (Modern):** Einen `record` verwenden. `Records` generieren korrekte `equals()`, `hashCode()` und `toString()`-Methoden automatisch.
- **Lösung B (IDE-Hilfe):** Die IDE (IntelliJ: `Alt`+`Einfg`) kann diese Methoden für normale Klassen automatisch und korrekt generieren.

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

- **Die `equals`-Checkliste:** Eine perfekte Implementierung prüft immer: 1. Referenz-Gleichheit (`this == o`), 2. `null`- und Typ-Gleichheit (`o == null || getClass() != o.getClass()`), 3. Cast und Feld-Vergleich.
- **`hashCode` generieren:** Die sicherste Art, einen `hashCode` aus mehreren Feldern zu generieren, ist die Hilfsmethode `Objects.hash(feld1, feld2, ...)`.
- **`toString()` und Instanzfelder:** Eine `toString()`-Methode sollte typischerweise nur den Zustand des Objekts (seine Instanzfelder) beschreiben, nicht die statischen Felder der Klasse.