Lernskript: Methoden-Überladung vs. Methoden-Überschreibung

Beide Konzepte klingen ähnlich, beschreiben aber fundamental unterschiedliche Mechanismen in Java.

Teil 1: Die grundlegende Analogie - Das "Warum"

- **Methoden-Überladung (`Overloading`) (Flexibilität):**
  - **Analogie:** Eine Lieferfirma bietet **verschiedene Wege an, dieselbe Aktion** (`auftragAnnehmen`) auszuführen, indem sie unterschiedliche Informationen akzeptiert (nur Adresse; Adresse + Express; etc.).
  - **Kernidee: Dieselbe Methode, aber mit unterschiedlichen Parameterlisten** (Anzahl, Typ oder Reihenfolge der Parameter). Findet **innerhalb einer Klasse** statt und dient der Bequemlichkeit.
- **Methoden-Überschreibung (`Overriding`) (Spezialisierung):**
  - **Analogie:** Eine `Drohne` führt eine Lieferung anders aus als ein normales `Fahrzeug`, obwohl die grundlegende Aktion `lieferungAusfuehren()` dieselbe ist.
  - **Kernidee:** Eine Subklasse liefert eine **neue Implementierung** für eine Methode, die bereits in der Superklasse existiert. Die **Methodensignatur (Name und Parameterliste) muss exakt identisch sein**. Findet im Kontext der **Vererbung** statt und ist die Grundlage für Polymorphie.

Teil 2: Wichtige Regeln & Vertiefung (Profi-Tipps)

- **Die Signatur-Regel (Der Kernunterschied):**
  - `Overloading`: Gleicher Name, **UNTERSCHIEDLICHE** Parameterliste.
  - `Overriding`: Gleicher Name, **EXAKT GLEICHE** Parameterliste.
- **Die `@Override`-Annotation:**
  - **Zweck:** Dient als **Sicherheitsnetz**. Du teilst dem Compiler deine Absicht mit, eine Methode zu überschreiben.
  - **Nutzen:** Wenn du dich bei der Signatur vertippst, würde der Compiler ohne `@Override` fälschlicherweise eine neue, überladene Methode erstellen. Mit `@Override` gibt der Compiler einen Fehler aus und weist dich auf den Tippfehler hin. **Benutze sie IMMER beim Überschreiben.**
- **Compile-Time vs. Run-Time (Statischer vs. Dynamischer Polymorphismus):**
  - **Überladung** wird zur **Compile-Zeit** aufgelöst. Der Compiler weiß anhand der übergebenen Argumente bereits, welche Methode er aufrufen muss.
  - **Überschreibung** wird zur **Laufzeit** aufgelöst. Erst zur Laufzeit wird anhand des tatsächlichen Objekttyps entschieden, welche überschriebene Methode (die der Super- oder Subklasse) ausgeführt wird.