V7 Fahrzeug

Vererbung

vererbung ist ein Prinzip der objektorientierten Programmierung, bei dem eine Klasse Eigenschaften und Methoden einer anderen Klasse übernimmt.

Warum braucht man Vererbung?

Es ist nicht nur wichtig zu wissen, **wie** Vererbung funktioniert, sondern auch **warum** sie verwendet wird. Hier betrachten wir einige Situationen, die die Notwendigkeit der Vererbung verdeutlichen:

1. **Code-Wiederverwendbarkeit**: Stellen wir uns vor, wir benötigen zusätzlich zur Auto-Klasse noch Klassen für Bus, LKW und Spielzeug-Auto. Ohne Vererbung müssten wir die komplette Arbeit der Auto-Klasse erneut für jede neue Klasse wiederholen. Vererbung ermöglicht es, gemeinsame Attribute und Methoden in einer Superklasse zu definieren. Die Subklassen können diese Attribute und Methoden erben, sodass wir den gemeinsamen Code nur **einmal** schreiben müssen.

2. **Erweiterbarkeit**: Wenn neue Attribute oder Methoden benötigt werden, können sie einfach zur Superklasse hinzugefügt werden. Dadurch erben alle Subklassen automatisch die neuen Funktionen, ohne dass man Änderungen in jeder einzelnen Klasse vornehmen muss.

3. **Einfaches Testen und Warten**: Gemeinsame Methoden müssen nur **einmal** in der Superklasse getestet werden. Dies macht die Wartung des Codes einfacher und reduziert den Aufwand für Fehlerbehebung.

4. **Saubere und klare Struktur**: Vererbung schafft eine klare Hierarchie und Struktur im Code, die die Lesbarkeit verbessert und die Entwicklung effizienter macht.

Situation Erklärung

Wir wollen eine **Auto**-Superklasse erstellen und davon die Klassen **Bus**, **LKW** und **Spielzeug-Auto** ableiten. Die Attribute wie:

- **Material**
- **Minimale und maximale Insassenzahl**
- **Baujahr**
- **Reifenradius**
- **Reifenbreite**
- **Marke**
- **Energietyp**

sind in allen Klassen gleich oder ähnlich und können deshalb in der Superklasse definiert werden.

In den Subklassen können spezifische Attribute hinzugefügt werden, wie zum Beispiel:

- **Batteriegröße** und **Anzahl** in der **Spielzeug-Auto-Klasse**.
- Ein **boolischer Wert** für eine **doppelte Etage** im **Bus**.

Auf diese Weise werden gemeinsame Eigenschaften in der Superklasse erstellt, während spezialisierte Attribute in den jeweiligen Subklassen definiert werden.

Methoden-Inheritance

Bei der Vererbung werden die Methoden der Superklasse unverändert auf die Subklassen übertragen. Das kann manchmal problematisch sein. Zum Beispiel:

- Die **fahr-Methode** der Auto-Klasse benötigt einen **Fahrer**, aber keine Batterien.
- Hingegen braucht das **Spielzeug-Auto** Batterien, jedoch keinen Fahrer.

Manchmal sind die Unterschiede zwischen den Subklassen noch größer. Deshalb benötigen wir eine Möglichkeit, Methoden in verschiedenen Subklassen anzupassen. Hier kommt die **Methodenüberschreibung** ins Spiel.

Methodenüberschreibung

In Dart überschreibt man eine Methode, indem man das **@override**-Symbol vor die Methode setzt. Dies ist nicht zwingend vom Compiler gefordert, sondern dient dazu, andere Entwickler darauf hinzuweisen, dass die Methode der Superklasse überschrieben wird. Anschließend definiert man die Methode einfach neu.

```
  @override
  void fahren() {
    if(baterienAnzahl == 0){
      print('Spielzeug kann nicht fahren, weil keine Batterien da sind');
      return;
    }
    print('Spielzeug fährt');
  }
```

Vererbung: `extends` und `implements`

In Dart gibt es zwei Hauptarten der Vererbung: `extends` und `implements`.

`extends`

Mit `extends` erbt die Subklasse die Eigenschaften und Methoden der Superklasse. Die Subklasse kann diese Methoden entweder übernehmen oder überschreiben, je nach Bedarf. Im Gegensatz dazu ist man bei der Implementierung gezwungen, alle Methoden und Attributen zu überschreiben.

Extends Code

```
class AutoV7 extends V7Fahrzeug {
  AutoV7({
    super.reifenZahl = 4,
    super.maxInsasseZahl = 5,
    required super.baujahr,
    super.reifenRadius = 30,
```

```
    super.reifenBreite = 22,
    required super.fahrer,
  });
  @override
  void fahren() {
    super.fahren();
    print('AutoV7 fährt');
  }
}
```

Konstruktor bei Vererbung mit `extends`

Bei der Vererbung mit `extends` müssen die Attribute, die in der **Superklasse** definiert sind, **nicht erneut** in der Subklasse definiert werden. Sie sind automatisch in der Subklasse verfügbar, da die Subklasse die Attribute und Methoden der Superklasse erbt.

Im Gegensatz zu normalen Attributen, die in der Subklasse verfügbar sind, wird beim Aufruf des Konstruktors ein spezielles Vorgehen verwendet. Die Attribute der Superklasse können im Konstruktor der Subklasse **nicht** direkt mit `this` zugewiesen werden, da sie in der Superklasse definiert sind. Stattdessen verwendet man das `super`**.Schlüsselwort**, um den Konstruktor der Superklasse aufzurufen und die Werte zuzuweisen.

Super Konstruktor mit Standardwerten

Falls man die Super-Attribute beim Erstellen des Objekts nicht sichtbar machen will, sondern einen Standardwert haben möchte, z. B. ein Spielzeugauto hat natürlich keinen Fahrer oder Insassin, sollte man in diesem Fall einen Super-Konstruktor verwenden, der alle zu verbergenden Attribute enthält, wie im folgenden Beispiel:

```
Spielzeug({
  required this.baterienGrosse,
  required this.baterienAnzahl,
  required super.baujahr,
  super.reifenRadius = 2,
  super.reifenBreite = 1,
}) : super(
        maxInsasseZahl: 0,
        reifenZahl: 4,
        fahrer: null,
      );
```

Aufgabe

Erstellen Sie eine **Motorrad-Klasse**, die von der **Fahrzeug-Klasse** erbt. Achten Sie dabei darauf, dass **nicht alle Attribute** der Fahrzeug-Klasse auch in der Motorrad-Klasse-Konstruktor erscheinen darf.