# V12 Fahrzeug

## Immutable Klass

Eine **immutable Klasse** (unveränderliche Klasse) ist eine Klasse, deren Instanzen nach der Erstellung nicht mehr verändert werden können. Das bedeutet, dass alle Felder dieser Klasse als `final` deklariert sind und nach der Initialisierung nicht mehr geändert werden dürfen.

### Eigenschaften einer immutable Klasse

1. **Finale Felder:** Alle Felder der Klasse müssen als `final` gekennzeichnet werden, damit sie nur einmal beim Erstellen des Objekts gesetzt werden können.
2. **Keine Setter:** Es gibt keine Methoden oder Wege, um die Werte der Felder nach der Objekterstellung zu ändern.
3. **Konstruktor:** Der Konstruktor setzt alle Werte der finalen Felder einmalig bei der Objekterstellung.

### Vorteile von immutable Klassen

1. **Thread-Sicherheit:** Da Objekte nicht verändert werden können, sind sie automatisch threadsicher. Es besteht keine Gefahr, dass ein Objekt von mehreren Threads gleichzeitig verändert wird.
2. **Verständlichkeit und Zuverlässigkeit:** Unveränderliche Objekte sind leichter zu verstehen, da man sicher sein kann, dass sich der Zustand eines Objekts nach der Initialisierung nicht ändert. Dies kann das Debugging und die Wartung des Codes erleichtern.
3. **Effizienz:** Immutable Objekte können effizienter verwendet werden, da sie als Konstanten zwischengespeichert werden können, wenn sie häufig verwendet werden. Dies führt zu einer besseren Leistung in einigen Situationen, z. B. bei der Verwendung als Schlüssel in HashMaps. **Beispiel**

```
class V12AutoImmutable {
 // Alle Attribute sind final
 final int reifenZahl;
 final int insasseZahl;
 final DateTime baujahr;
  // 28 => 38
 final double reifenRadius;
  // 18 => 25
 final double reifenBreite;
 final String? marke;
 // Konstruktor ist immutable (const).
 const V12AutoImmutable({
    this.reifenZahl = 4,
    this.insasseZahl = 5,
    required this.baujahr,
    this.reifenRadius = 30,
    this.reifenBreite = 20,
    this.marke,
  });
  }
```

# Copy-With

Es ist Methode eine Funktion, die in Dart verwendet wird, um eine Kopie eines Objekts zu erstellen, wobei bestimmte Felder aktualisiert werden können, während die anderen Felder unverändert bleiben.

# Warum Copy-With?

Immutable Klassen bieten viele Vorteile, doch sie ermöglichen es nicht, bereits erstellte Objekte zu ändern. Dies kann in einigen Situationen unpraktisch sein. Um dieses Problem zu lösen, wurde das **Copy-With**-Muster entwickelt.

### Beispiel zur Veranschaulichung

Stellen Sie sich vor, Sie möchten einen Antrag ausfüllen, haben jedoch mit einem Kuli einen Fehler gemacht und kein Werkzeug, um ihn zu löschen. In diesem Fall würden Sie Ihren Antrag wegwerfen und einen neuen ausfüllen. Genau das macht das Copy-With-Muster: Anstatt die Werte der Attribute eines bestehenden Objekts zu ändern, erstellt es ein neues Objekt mit den gewünschten Attributen und ersetzt das alte Objekt durch das neue.

### Vorteile von Copy-With

1. **Unveränderlichkeit:** Es bleibt bei der Unveränderlichkeit, da das ursprüngliche Objekt unberührt bleibt.
2. **Einfache Anpassungen:** Sie können nur die Werte ändern, die Sie benötigen, während der Rest des Objekts unverändert bleibt.
3. **Vermeidung von Nebeneffekten:** Es gibt keine unerwarteten Änderungen am ursprünglichen Objekt, was die Nachvollziehbarkeit und Wartbarkeit des Codes verbessert.
4. **Einfacher Validierung:** Die Validierung der erneuten Werte kann einfach innerhalb der copyWith-Methode durchgeführt werden, ohne dass Setters und Getters benötigt werden.

# Copy-With Bauschritte

Die Copy-With-Methode ist nichts anderes als eine normale Methode, die ein Objekt ihrer Klasse zurückgibt. Daher ist der Rückgabewert der Copy-With-Methode immer vom Typ der Klasse.

### Eingabeparameter

- **Named Parameter:** Die Copy-With-Methode verwendet ein Named-Parameter-Feld, da die Eingaben oft zahlreich sind und normale unnamed Parameter komplex und fehleranfällig sein können.
- **Nullable Parameter:** Alle Parameter im Eingabefeld sollten nullable sein. Dadurch kann man nur die Werte angeben, die man ändern möchte, während die anderen Parameter unverändert bleiben.

### Implementierung

1. **Attribute:** In den Eingabeparametern kommen alle Attribute der Klasse vor, außer denjenigen, die nach der ersten Erstellung des Objekts nicht mehr geändert werden dürfen (z.B. `baujahr` in unserem Beispiel).

2. **Zuweisung:** Innerhalb der geschweiften Klammern der Copy-With-Methode werden die nicht-nullbaren Parameter in den Eingabefeldern ihren Werten zugewiesen. Für die Attribute, die nicht

geändert werden, bleibt der ursprüngliche Wert erhalten. Dies gewährleistet, dass nur die angegebenen Parameter in den Eingabefeldern sich ändern können.

### Hinweis

- **Keine eingebaute Methode:** Copy-With ist keine eingebaute Methode in Dart.
- **Alternativer Name:** Die Copy-With-Methode kann auch einen anderen Namen erhalten, aber es ist eine allgemeine Konvention, sie so zu nennen.

```
// CopyWith definition
// Named Parameter
 V12AutoImmutable copyWith({
  // Alle Attribute sind nullable
  // Baujahr ist entfernt, weil es darf logischerweise nicht geändert werden.
    int? reifenZahl,
    int? insasseZahl,
    double? reifenRadius,
    double? reifenBreite,
    String? marke,
  }) {
    return V12AutoImmutable(
      // Wenn der Wert null ist, wird der aktuelle Wert verwendet.
      reifenZahl: reifenZahl ?? this.reifenZahl,
      insasseZahl: insasseZahl ?? this.insasseZahl,
      // baujahr ist nicht veränderbar( haltet ihre Wert zu)
      baujahr:  baujahr,
      reifenRadius: reifenRadius ?? this.reifenRadius,
      reifenBreite: reifenBreite ?? this.reifenBreite,
      marke: marke ?? this.marke,
    );
  }
```

---

## Aufgabe

Mensch-Klasse: Machen Sie Ihre Mensch-Klasse immutable und erstellen Sie eine copyWith-Methode dafür. Hinweis: Einige Attribute in der Mensch-Klasse dürfen auch mit copyWith nicht geändert werden.