V14 Fahrzeug

ToMap und FromMap für komplexe Attribute

In der letzten Lektion haben wir die `toMap` und `fromMap` Methoden mit einfachen Attributen, die bereits in Dart eingebaut sind, behandelt. Das hat gut funktioniert, weil die eingebauten JSON-Funktionen wie `jsonEncode` und `jsonDecode` diese einfachen Datentypen verarbeiten können.

Problemstellung bei komplexen Objekten

Wenn aber Attribute komplexe Typen haben, die aus Klassen bestehen (z.B. Klassenobjekte), können die eingebauten JSON-Funktionen diese nicht direkt verarbeiten. JSON kann nur einfache Datentypen wie Strings, Zahlen, Listen und Mappen erkennen und serialisieren.

Lösung

Um dieses Problem zu lösen, müssen wir die komplexen Objekt-Attribute in einfache darstellbare Typen umwandeln. Das kann erreicht werden, indem man jedes Objekt-Attribut in eine `Map<String, dynamic>` umwandelt, bevor es in JSON serialisiert wird. Dies bedeutet, dass jedes Klassen-Objekt, das ein Attribut ist, eigene `toMap` und `fromMap` Methoden haben muss, um die JSON-Funktionen korrekt zu unterstützen.

Beispiel

```
// die gleiche Methode wie in letzter Lektion
  Map<String, dynamic> toMap() {
    return <String, dynamic>{
      'reifenZahl': reifenZahl,
      'insasseZahl': insasseZahl,
      'baujahr': baujahr.millisecondsSinceEpoch,
      'reifenRadius': reifenRadius,
      'reifenBreite': reifenBreite,
      'marke': marke,
      // engine ist ein Objekt, das in eine Map umgewandelt wird
      // somit kann die json.encode Methode das Objekt serialisieren
      'engine': engine.toMap(),
    };
  }
// die gleiche Methode wie in letzter Lektion
  factory V14Auto.fromMap(Map<String, dynamic> map) {
    return V14Auto(
      reifenZahl: map['reifenZahl'] as int,
      insasseZahl: map['insasseZahl'] as int,
      baujahr: DateTime.fromMillisecondsSinceEpoch(map['baujahr'] as int),
      reifenRadius: map['reifenRadius'] as double,
      reifenBreite: map['reifenBreite'] as double,
      marke: map['marke'] != null ? map['marke'] as String : null,
      // engine ist ein Objekt, das aus einer Map erstellt wird
      // somit kann die json.decode Methode das Objekt deserialisieren
      engine: Engine.fromMap(map['engine'] as Map<String, dynamic>),
```

```
    );
  }
```

Kurzgefasst

Die `Engine`-Klasse ist kein einfacher Datentyp, sondern eine Klasse. Daher muss sie, wenn sie als Attribut in einer anderen Klasse (`Auto`) verwendet wird, eigene `toMap` und `fromMap` Methoden besitzen. Diese Methoden sind notwendig, um die `Engine`-Objekte innerhalb der `toMap` und `fromMap` Methoden der `Auto`-Klasse korrekt zu verarbeiten.

Aufgabe

Erstellen Sie eine **`Mensch-Klasse`** mit den Attributen **`name`**, **`vorname`**, **`geburtsjahr`** und einem **`Mund-Objekt`**. Dieses Mund-Objekt soll die Attribute double **`breite`**, int **`zahnanzahl`** und int **`defektezahnAnzahl`** enthalten.

Erstellen Sie für die Mensch-Klasse die Methoden **`toMap`**, **`fromMap`**, **`toJson`** und **`fromJson`**. Achten Sie darauf, dass Mund ein **komplexes** Objekt ist.