Enum

Ein Enum (Enumeration) ist eine spezielle Datentyp-Definition, die es ermöglicht, eine festgelegte Anzahl von Werten zu definieren, aus denen der Benutzer wählen kann. Enums erleichtern die Verwendung und Lesbarkeit von Code, indem sie symbolische Namen für vordefinierte Werte bereitstellen.

Es gibt zwei grundlegende Typen von Enums in Dart:

1. **Simple Enum**: Ein einfacher Enum erlaubt dem Benutzer, zwischen begrenzten Varianten zu wählen und eignet sich, wenn keine weiteren Informationen für die Auswahl benötigt werden. Zum Beispiel könnte ein `Motor` die Varianten `Benzin`, `Diesel`, `Gas`, `Elektrizität` oder `Hybrid` haben. Wenn diese Auswahl lediglich die Art des Antriebs darstellt und keine weiteren Daten gespeichert werden müssen, reicht ein einfacher Enum.

   Sollte der Aufbau des Motors jedoch von der gewählten Antriebsart abhängen oder sollen spezifische Informationen zu jedem Typ gespeichert werden (z.B. spezifische Leistungswerte oder Verbrauchswerte), dann kann ein Simple Enum unpraktisch sein. Es müsste dann zusätzlich eine Reihe von `if`-Bedingungen im Code verwendet werden, um die Unterschiede zwischen den Varianten abzufragen, was den Code unübersichtlich machen kann.

```
   class Engin{
    EnginEnum motor;
    String name;
    Engin(this.motor, this.name);

    Map<String, dynamic> toMap() {
      // motor.toString()=> EnginEnum.diesel. und das will man nicht. sondern nur
'Diesel'
      // deshalb wird ein Switch-Case verwendet. und das ist zu viel Code. nur für
toMap-Methode
      String motorType;
      switch (motor) {
        case EnginEnum.diesel:
          motorType = 'Diesel';
          break;
        case EnginEnum.benzin:
          motorType = 'Benzin';
          break;
        case EnginEnum.elektro:
          motorType = 'Elektro';
          break;
        case EnginEnum.gas:
          motorType = 'Gas';
          break;
        case EnginEnum.hybrid:
          motorType = 'Hybrid';
          break;
        default:
          motorType = 'Unbekannt';
```

```
      }
      return {
        'motor': motorType,
        'name': name,
      };
    }
        int get tankenKosten {
      switch (motor) {
        case EnginEnum.diesel:
          return 50;
        case EnginEnum.benzin:
          return 60;
        case EnginEnum.elektro:
          return 30;
        case EnginEnum.gas:
          return 40;
        case EnginEnum.hybrid:
          return 45;
        default:
          return 0;
       }
     }

  }
```

2. **Class Enum**: Um die Einschränkungen eines einfachen Enums zu überwinden, kann man das Klass-Enum verwenden. Ein Klass-Enum erlaubt die Erstellung von standardisierten Objekten, die nicht nur die Auswahlvarianten enthalten, sondern auch spezifische Attribute und Methoden definieren können.

   Durch die Verwendung von Klass-Enums kann man jedem Enum-Wert (z.B. `Benzin`, `Diesel`, `Elektrizität`) zusätzliche Eigenschaften wie `leistung`, `verbrauch`, oder andere spezifische Parameter zuweisen. Auf diese Weise werden alle erforderlichen Daten direkt im Enum gespeichert, sodass man auf zusätzliche `if`-Abfragen verzichten kann. Ein Klass-Enum bietet dadurch eine strukturierte Lösung, die den Code klarer und wartbarer macht, besonders in Fällen, in denen sich das Verhalten oder die Daten je nach Auswahl ändern.

Ein Klass-Enum bietet somit mehr Flexibilität und ist sinnvoll, wenn zusätzliche Informationen erforderlich sind.

```
enum EnginEnumMitStringElemente {
  // jeweils Elemente gilt als Objekt mit einem String-Label
  diesel('Diesel',50),
  benzin('Benzin',60),
  elektro('Elektro',30),
  gas8('Gas',40),
  hybrid('Hybrid',45),
  ;

  // wie eine normale Klasse, die Attribute und Komstruktor hat.
  final String lable;
  final int tankenKosten;
```

```
  const EnginEnumMitStringElemente(this.lable, this.tankenKosten);
}

class AutoMitStringEnum {
  EnginEnumMitStringElemente motor;
  String name;
  AutoMitStringEnum(this.motor, this.name);
 // die Enum Klasse hat den Bau von Methoden viel einfacher gemacht.
  Map<String, dynamic> toMap() {
    return {
      'motor': motor.lable,
      'name': name,
    };
  }

  int get tankenKosten {
    return motor.tankenKosten;
  }

}
```

Situationsbeschreibung

Wir möchten eine `Fahrzeug`-Enum-Klasse erstellen, die drei spezifische Fahrzeugobjekte repräsentiert: `Auto`, `Bus` und `LKW`. Jedes Objekt hat drei Attribute: `name`, `geschwindigkeit` und `sitzplatzAnzahl`.

Bauschritte zur Erstellung der Enum-Klasse

1. **Normale Enum-Erstellung**:
   Erstelle eine Enum-Klasse `Fahrzeug` mit den Elementen `Auto`, `Bus` und `LKW`.

2. **Semikolon hinzufügen**:
   Nach der Auflistung der Elemente `Auto`, `Bus` und `LKW` füge innerhalb der geschweiften Klammern ein Semikolon (`;`) hinzu, um anzuzeigen, dass die Enum zusätzliche Definitionen enthält.

3. **Attribute definieren**:
   Definiere die benötigten Attribute: `name`, `geschwindigkeit`, und `sitzplatzAnzahl`.

4. **Konstruktor erstellen**:
   Erstelle einen Konstruktor, der diese Attribute initialisiert.

5. **Default-Werte festlegen**:
   Weise jedem Enum-Element (`Auto`, `Bus`, `LKW`) Standardwerte für `name`, `geschwindigkeit` und `sitzplatzAnzahl` zu, indem du sie im Konstruktor initialisierst.

Codebeispiel

Hier ist ein Beispiel, wie die `Fahrzeug`-Enum-Klasse aussehen könnte:

```
enum FahrzeugEnum {
// Hier wird die Enum-Klasse Elemente definiert.
  auto(geschwindigkeit: 220, name: 'BMW', sitzplaetze: 5),
  bus(geschwindigkeit: 180, name: 'Mazedis', sitzplaetze: 5),
  lkw(geschwindigkeit: 120, name: 'Toyota', sitzplaetze: 2),
  ;

  // Attribute
  final String name;
  final int geschwindigkeit;
  final int sitzplaetze;
  // Konstruktor neim Eunm Klasse kann nur Const sein.
  const FahrzeugEnum({required this.name, required this.geschwindigkeit, required
this.sitzplaetze});

// Enum Klasse kann auch Getters und Methoden haben.
  String get info => 'FahrzeugEnum: name: $name, Geschwindigkeit:
$geschwindigkeit, Sitzplätze: $sitzplaetze';

  void fahren() {
    // Die Methode können spezifiziert werden.
    if (this == FahrzeugEnum.auto) print('Das Auto fährt');
    if (this == FahrzeugEnum.bus) print('Der Bus fährt');
    if (this == FahrzeugEnum.lkw) print('Der LKW fährt');
  }

  void bremsen() {
    print('Das Fahrzeug bremst');
  }
}
```

Wichtige Hinweise zu Enums

1. **Konstruktor gehört den Objekt-Elementen**:
   Der Konstruktor innerhalb eines Enums gehört nicht dem Enum selbst, sondern den vordefinierten Objekt-Elementen (z.B. `Auto`, `Bus`, `LKW`). Daher kann der Konstruktor nicht direkt aufgerufen werden. Ein Versuch wie `final auto2 = Fahrzeug();` führt zu einem Fehler.

2. **Enums enthalten nur vordefinierte Objekte**:
   Eine Enum-Klasse kann nur vordefinierte, unveränderliche Objekte enthalten. Man kann keine neuen Instanzen zur Laufzeit hinzufügen.

3. **Const-Konstruktor und final Attribute**:
   Der Konstruktor eines Enums muss als `const` definiert sein, und alle Attribute des Enums müssen `final` sein, um sicherzustellen, dass sie unveränderlich sind.

4. **Keine Factory-Methoden in Enums**:
   Eine Enum-Klasse kann keine `factory`-Methoden enthalten, da eine `factory` am Ende immer einen Konstruktor aufrufen muss. Da der Konstruktor im Enum aber den Objekt-Elementen gehört und nicht dem Enum selbst, ist dies nicht möglich.

5. **Methoden und Getter innerhalb von Enums**:
   Enums können benutzerdefinierte Methoden und Getter enthalten, was die Funktionalität erweitert und eine bessere Kontrolle über die Objekte ermöglicht.

6. **Statische Methoden und Attribute**:
   Es ist möglich, statische Methoden und Attribute in einer Enum-Klasse zu definieren. Diese können aufgerufen werden, ohne dass eine Instanz des Enums erstellt werden muss.

Trick zur Erweiterung von Enum-Fähigkeiten für benutzerdefinierte Objekte

Da Enums nur vordefinierte, unveränderliche Objekte enthalten können, ist es normalerweise nicht möglich, neue Objekte dynamisch zu erstellen. Mit einem Trick lässt sich jedoch eine ähnliche Funktionalität erreichen, indem man eine separate `Fahrzeug`-Klasse erstellt und eine statische Methode im Enum definiert, die eine Instanz dieser Klasse zurückgibt. So wirkt der Aufruf fast wie die Erstellung eines Objekts innerhalb des Enums.

Bau der Lösung

1. Erstelle eine `Fahrzeug`-Klasse mit den gleichen Attributen wie das Enum (`name`, `geschwindigkeit`, `sitzplatzAnzahl`).
2. Füge eine statische Methode im Enum `FahrzeugEnum` hinzu, die eine Instanz der `Fahrzeug`-Klasse zurückgibt.
3. Damit das Objekt richtig funktioniert, muss beim Aufruf der Methode das `as Fahrzeugname`-Schlüsselwort hinzugefügt werden, um den Typ explizit festzulegen.

```
enum FahrzeugEnum {
// Hier wird die Enum-Klasse Elemente definiert.
  auto(geschwindigkeit: 220, name: 'BMW', sitzplaetze: 5),
  bus(geschwindigkeit: 180, name: 'Mazedis', sitzplaetze: 5),
  lkw(geschwindigkeit: 120, name: 'Toyota', sitzplaetze: 2),
  ;
  // Attribute
  final String name;
  final int geschwindigkeit;
  final int sitzplaetze;
  // Konstruktor neim Eunm Klasse kann nur Const sein.
  const FahrzeugEnum({required this.name, required this.geschwindigkeit, required
this.sitzplaetze});
  // Enum können keinen spzifisch Bau des Objekts haben.
  // Enum können keinen Factory-Konstruktor haben.
  // als Alternative kann eine statische Methode verwendet werden, die ein Objekt
ähnlicher Klasse zurückgibt.
  static flexiibleAuto({required String name, required int geschwindigkeit,
required int sitzplaetze}) {
    return FlexibleAuto(name: name, geschwindigkeit: geschwindigkeit, sitzplaetze:
sitzplaetze);
  }

}
class FlexibleAuto {
```

```
  final String name;
  final int geschwindigkeit;
  final int sitzplaetze;
  FlexibleAuto({required this.name, required this.geschwindigkeit, required
this.sitzplaetze});
}
void main(){
      final flexibleAuto = FahrzeugEnum.flexiibleAuto(name: 'BMW-X6',
geschwindigkeit: 320, sitzplaetze: 5) as FlexibleAuto;
}
```

Aufgabe

Ziel Erstellen Sie ein Enum namens `ObstEnum`, das eine Liste verschiedener Obstsorten mit ihren Eigenschaften enthält. Jede Obstsorte soll über folgende Attribute verfügen:

- `name`: Der Name des Obsts.
- `farbe`: Die Farbe des Obsts.
- `geschmack`: Der Geschmack des Obsts (z.B. süß, sauer).

Zusätzlich soll das Enum eine statische Liste haben, die nur die Namen der Obst-Objekte enthält.