Static in Klassen

In Dart können Klassen `static` Methoden und Attribute enthalten. Diese `static`-Elemente gehören nicht zu den Instanzen (Objekten) der Klasse, sondern ausschließlich zur Klasse selbst. Das bedeutet, sie sind unabhängig von konkreten Objekten der Klasse und können direkt über die Klasse aufgerufen werden.

Warum Static-Elemente

1. **Direkter Zugriff ohne Instanz**: Da `static`-Elemente zur Klasse selbst gehören, kann man auf sie zugreifen, ohne ein Objekt der Klasse zu erstellen.
2. **Speicher- und Performancevorteile**: `Static`-Elemente existieren nur einmal im Speicher, unabhängig davon, wie viele Objekte der Klasse instanziiert werden.

Wichtige Hinweise zu Static-Elementen

1. **Zugriff nur über die Klasse**
   `Static`-Elemente können nur über die Klasse selbst und nicht über ihre Objekte zugewiesen oder aufgerufen werden.

2. **Datentypen für Static Attribute**
   Ein `static` Attribut kann `const`, `final` oder `var` sein, je nach der gewünschten Flexibilität und dem Anwendungsfall.

3. **Einschränkungen bei Static Methoden**
   `Static` Methoden können auf andere `static` Attribute und Methoden der Klasse zugreifen, jedoch nicht auf normale (nicht-`static`) Attribute oder Methoden, da diese an Objekte gebunden sind.

4. **Keine Vererbung von Static Elementen**
   `Static` Elemente werden nicht vererbt oder implementiert, da sie nur der Klasse selbst gehören, in der sie definiert sind. Sie sind nicht für Unterklassen oder implementierende Klassen verfügbar.

Verwendung von Static Elementen in Klassen

Static-Elemente sind vielseitig einsetzbar und bieten viele Vorteile für die Strukturierung und Vereinfachung des Codes. Hier sind einige typische Einsatzmöglichkeiten:

1. **Globale Konstante Variablen in einer Klasse kapseln**

Static-Attribute eignen sich hervorragend, um globale Konstanten in einer Klasse zu organisieren. Zum Beispiel könnte eine `UrlsKlass` sämtliche URL-Endpunkte einer Anwendung zentral verwalten:

```
class UrlsKlass {
  static const String baseUrl = 'http://www.example.com';
  static const String loginUrl = '/login';
  static const String registerURL = '/register';
  static const String profileURL = '/profile';
  static const String logoutURL = '/logout';
  static const String homeURL = '/home';
```

```
  static const String settingsURL = '/settings';
  static const String aboutURL = '/about';
}
```

2. **Globale Funktionen für ähnliche Aufgaben: Taschenrechner-Beispiel**

Mit static Methoden lassen sich Funktionen erstellen, die nicht an Instanzen der Klasse gebunden sind, sondern direkt aufgerufen werden können. Dies ist besonders nützlich für Werkzeuge wie einen Taschenrechner:

```
class TaschenRechner {
  static void addieren(int a, int b) {
    print('$a + $b = ${a + b}');
  }

  static void subtrahieren(int a, int b) {
    print('$a - $b = ${a - b}');
  }

  static void multiplizieren(int a, int b) {
    print('$a * $b = ${a * b}');
  }

  static void dividieren(int a, int b) {
    print('$a / $b = ${a / b}');
  }
}
```

3. **Attribute und Methoden, die für alle Objekte einer Klasse gelten: Anzahl der erstellten Objekte**

Static-Elemente eignen sich gut, um gemeinsame Eigenschaften oder Methoden für eine ganze Klasse zu speichern, z. B. die Anzahl der erstellten Objekte oder eine Liste aller Objekte der Klasse.

```
abstract class FahrzeugV23 {
  // Static Attribute: Anzahl der Fahrzeuge und Liste aller Fahrzeuge.
  static int fahrzeugeAnzahl = 0;
  static final List<FahrzeugV23> fahrzeuge = [];
  static const int reifenZahl = 4; // Konstante für alle Fahrzeuge
  int geschwindigkeit = 220; // Instanzattribut

  FahrzeugV23() {
    // Erhöht die Anzahl der Fahrzeuge bei jeder Instanziierung.
    fahrzeugeAnzahl++;
    // Fügt jedes neue Fahrzeug zur Liste hinzu.
    fahrzeuge.add(this);
  }
```

```
  // Static Methode: Funktionalität für alle Fahrzeuge, ohne auf Instanzattribute
zuzugreifen.
  static void fahren() {
    print('Fahrzeug fährt');
    // geschwindigkeit = 50; // Fehler: Auf Instanzattribute kann hier nicht
zugegriffen werden.
  }
}
```

Augabe

Erstellen Sie eine Rezept-Klasse, die eine `List<String>` für die Zutaten und eine `double` für die Vorbereitungsdauer enthält. Diese Klasse soll ein statisches List`<Rezept>`-Attribut haben, das alle Rezepte speichert. Zusätzlich soll eine statische Methode deleteRezept(int index) implementiert werden, die einen Index erhält und das entsprechende Rezept aus der statischen rezepte-Liste entfernt.