Operation Überschreibung

Operator-Überschreibung in Dart

In Dart kann man grundlegende Datentypen wie `int` oder `String` einfach mit dem `==` Operator vergleichen, um zu überprüfen, ob sie gleich sind. Bei Objekten verhält sich der Vergleich jedoch anders. Der `==` Operator vergleicht standardmäßig nur die Referenz der Objekte im Speicher und nicht deren Attributwerte. Daher werden zwei Objekte mit identischen Attributen als ungleich betrachtet, weil sie auf verschiedene Speicherorte zeigen.

Beispiel

```
class Auto {
  String marke;
  int baujahr;

  Auto({required this.marke, required this.baujahr});
}

void main() {
  Auto auto1 = Auto(marke: 'Audi', baujahr: 2020);
  Auto auto2 = Auto(marke: 'Audi', baujahr: 2020);

  print(auto1 == auto2); // Gibt false zurück, weil die Referenzen unterschiedlich
sind
}
```

In diesem Beispiel sind auto1 und auto2 zwar in ihren Attributen gleich, werden aber aufgrund ihrer unterschiedlichen Speicherreferenzen als ungleich betrachtet.

Lösung

Der Operator == überschreiben Um zwei Objekte als gleich zu betrachten, wenn ihre Attributswerte übereinstimmen, kann man den == Operator überschreiben und die Attributwerte manuell vergleichen. **Wichtiger Hinweis:** Wenn man den Vergleichsoperator (==) überschreibt, muss man auch den **hashCode** überschreiben.

Warum? Der hashCode wird verwendet, um Objekte in Sammlungen wie HashSet oder HashMap **effizient zu speichern**. Wenn der ==-Operator überschrieben wird, müssen Objekte mit gleichen Attributen auch denselben hashCode haben. Dadurch wird sichergestellt, dass sie in solchen Sammlungen korrekt als identisch behandelt werden.

Vergleichsoperator-Überschreibung: Schritt-für-Schritt-Anleitung

Beim Überschreiben des Vergleichsoperators `==` in Dart gehen wir in mehreren Schritten vor, um den Vergleich für benutzerdefinierte Objekte korrekt zu implementieren.

Bauschritte

1. `@override` **schreiben:**
    - Dies signalisiert, dass die Methode von der Oberklasse überschrieben wird. Es ist eine gute Praxis, um sicherzustellen, dass die Methode korrekt überschrieben wird und andere Entwickler gewarnt sind.
2. **Den Rückgabetyp definieren:**
    - Der Rückgabetyp des Vergleichs ist `bool`, da der Vergleich entweder `true` oder `false` zurückgeben soll.
3. **Den Operator festlegen:**
    - In diesem Fall überschreiben wir den `==` Operator. Die Syntax lautet: `operator ==`.
4. **Das Eingabefeld bauen:**
    - Verwende `covariant` im Parameter, um sicherzustellen, dass das übergebene Objekt von einem kompatiblen Typ ist. Danach kommt der Typ des zu vergleichenden Objekts und sein Name (in der Regel `other`).
5. **Vergleichslogik in geschweiften Klammern definieren:**
    - Definiere die Regeln, nach denen die Objekte verglichen werden sollen. Vergleiche die relevanten Attribute der Objekte.

Beispiel

```
 @override
bool operator ==(covariant V17Auto other) {
  // Überprüft, ob beide Objekte identisch im Speicher sind.
  // Falls ja, sind sie gleich.
  // mögliche kürzerer Weg: return true;
  if (identical(this, other)) {
    print('identical');  // Gibt aus, dass die Objekte identisch sind.
    return true;
  }
  // Wenn die Objekte nicht identisch sind, wird geprüft, ob
  // die relevanten Eigenschaften der Objekte gleich sind.
  print('not identical');  // Gibt aus, dass die Objekte nicht identisch sind.
  return other.maxGeschwindigkeit == maxGeschwindigkeit &&
         other.maxLeistung == maxLeistung &&
         other.maxInsasssenAnzahl == maxInsasssenAnzahl;
}
```

Hashcode-Überschreibung Bauschritte

1. `@override` **schreiben:**

   - Dies zeigt an, dass wir die `hashCode` Methode der Oberklasse überschreiben.

2. **Den Rückgabetyp definieren:**

   - Der `hashCode` gibt immer einen `int` zurück, der die Objektinstanz repräsentiert.

3. **Hashcode-Regeln definieren:**

   - Wir berechnen den `hashCode` basierend auf den Attributen, die wir auch im `==` Operator verwenden. Es gibt mehrere Möglichkeiten, den `hashCode` zu berechnen:
   - Die einfachste Möglichkeit besteht darin, den `hashCode` der Attribute zu kombinieren.
   - Verwende den `^` Operator (bitweises XOR) oder die Methode `Object.hash()`.
   - **Wichtiger Hinweis** bei der Überschreibung des hashCode: Objekte, die mit der neuen Vergleichsmethode (==) als gleich betrachtet werden, müssen denselben hashCode haben. Das bedeutet, dass alle Attribute, von denen die neue Vergleichslogik abhängt, auch in die Berechnung des hashCode einbezogen werden müssen. Nur so wird sichergestellt, dass Objekte, die als gleich angesehen werden, auch denselben hashCode besitzen und korrekt in Sammlungen wie HashSet oder HashMap behandelt werden.

Beispiel

```
// Die Annotation @override kennzeichnet, dass die Methode "hashCode"
// die Standardimplementierung aus einer übergeordneten Klasse oder Schnittstelle
überschreibt.
// Berechnet den Hashcode für das V17Auto-Objekt.
// Der Hashcode wird aus den Eigenschaften maxGeschwindigkeit, maxLeistung und
maxInsasssenAnzahl berechnet,
// um die Gleichheit der Objekte effizient zu überprüfen.
@override
int get hashCode =>
    maxGeschwindigkeit.hashCode ^ // Hashcode der maxGeschwindigkeit
    maxLeistung.hashCode ^        // Hashcode der maxLeistung
    maxInsasssenAnzahl.hashCode;  // Hashcode der maxInsasssenAnzahl
```

Überschreibung von anderen Operatoren

Neben dem `==`-Operator können in Dart auch andere Operatoren wie `+`, `-`, `*` oder `/` überschrieben werden, um Klassen benutzerdefinierte Funktionalität zu geben. Anders als bei der `==`-Überschreibung ist es jedoch nicht notwendig, den `hashCode` neu zu definieren, da es sich um rein arithmetische Operationen handelt.

Situationserklärung

Stellen wir uns eine Klasse `Land` vor, die zwei Attribute besitzt:

- `einwohnerAnzahl`: Ein `int`, der die Anzahl der Einwohner eines Landes repräsentiert.
- `flaeche`: Ein `double`, das die Fläche des Landes in Quadratkilometern angibt.

Wenn zwei Länder fusionieren, müssen sowohl die Einwohneranzahl als auch die Fläche der beiden Länder zusammengezählt werden. Dies erreichen wir, indem wir den `+`-Operator für die Klasse `Land` überschreiben.

Dadurch können wir zwei Länder-Objekte einfach mit dem `+`-Operator kombinieren, und die Summe der Einwohner sowie der Flächen wird berechnet.

Zusätzlich gibt es eine `Union`-Klasse, die ein einziges Attribut besitzt: eine Liste von Ländern (`List<Land>`). Wenn man zwei Länder miteinander multipliziert, wird eine Union dieser beiden Länder erstellt.

Die Klasse `Union` hat auch eine eigene `+`-Operation. Diese erlaubt es, ein weiteres Land zur Union hinzuzufügen oder sogar zwei Unions miteinander zu kombinieren, indem man ihre Listen von Ländern zusammenführt.

Hinweise zur Überschreibung von Operatoren

1. **Ähnlichkeit zu Settern**: Die Überschreibung eines Operators sieht sehr ähnlich wie die Implementierung eines Setters aus. Der Hauptunterschied besteht darin, dass man das Schlüsselwort `operator` anstelle von `set` verwendet, und anstelle eines Setternamens wird das Symbol des gewünschten Operators (wie `+`, `-`, `*` usw.) genutzt.
2. **Kein @override erforderlich**: Bei der Überschreibung von Operatoren, mit Ausnahme von `==`, benötigt man kein `@override`-Schlüsselwort, da diese Operatoren in der Regel nicht in der Klasse definiert sind.
3. **Gültigkeit**: Die Überschreibung einer Operation ist nur für die Objekte dieser Klasse gültig.
4. **Ergebnis-Typ**: Das Ergebnis einer Operator-Überschreibung muss nicht zwingend vom Typ der Klasse sein, in der die Überschreibung erfolgt. Das bedeutet, dass Sie einen Operator so definieren können, dass er einen anderen Datentyp zurückgibt.
5. **Argumenttyp**: Der andere Teil der Operation muss ebenfalls nicht unbedingt vom gleichen Typ wie die Klasse sein. Sie können Operatoren so überschreiben, dass sie mit verschiedenen Datentypen arbeiten, je nach den Anforderungen Ihrer Anwendung.

Beispiel

```
class Land {
  int einwohnerAnzahl;
  double flaeche;

  Land({required this.einwohnerAnzahl, required this.flaeche});

  // Überschreibe den + Operator
  Land operator +(Land other) {
    return Land(
      einwohnerAnzahl: this.einwohnerAnzahl + other.einwohnerAnzahl,
      flaeche: this.flaeche + other.flaeche,
    );
  }

  // beim Multiplizieren von zwei Ländern wird ein Union erstellt
  Union operator *(Land other) {
    return Union(laender: [this, other]);
  }
```

```
  @override
  String toString() {
    return 'Land: Einwohner: $einwohnerAnzahl, Fläche: $flaeche km²';
  }
}

class Union {
  List<Land> laender;
  Union({required this.laender});

  int get einwohnerAnzahl => laender.map((land) =>
land.einwohnerAnzahl).reduce((a, b) => a + b);

  double get flaeche => laender.map((land) => land.flaeche).reduce((a, b) => a +
b);

  @override
  String toString() {
    return 'Union: Einwohner: $einwohnerAnzahl, Fläche: $flaeche km²';
  }

  Union operator +(dynamic other) {
    if (other.runtimeType == Land) {
      return Union(laender: laender..add(other));
    }
    if (other.runtimeType == Union) {
      return Union(laender: [...laender]..addAll(other.laender));
    }
    throw Exception('Invalid type, Operator darf nur mit Land oder Union verwendet
werden');
  }
}

void main() {
  Land deutschland = Land(einwohnerAnzahl: 83000000, flaeche: 357022);
  Land oesterreich = Land(einwohnerAnzahl: 8900000, flaeche: 83879);
  Land denmark = Land(einwohnerAnzahl: 5800000, flaeche: 42952);
  // Die Ausgabe ist ein Land
  Land vereinigteLaender = deutschland + oesterreich;
  // Die Ausgabe ist ein Union
  Union union = deutschland * oesterreich;

  print(vereinigteLaender); //Land: Einwohner: 91900000, Fläche: 440901.0 km²
  print(union); // Union: Einwohner: 91900000, Fläche: 440901.0 km²
  union = union + denmark;
  print(union); // Union: Einwohner: 97700000, Fläche: 483853.0 km²
  union = union + union;
  print(union); // Union: Einwohner: 195400000, Fläche: 967706.0 km²
}
```

Aufgabe

Stellen wir uns eine Klasse `Mensch` vor, die wichtige Attribute enthält wie:

- `name`: Ein `String`, der den Namen einer Person repräsentiert.
- `alter`: Ein `int`, der das Alter einer Person beschreibt.
- `beruf`: Ein `String`, der den Beruf der Person angibt.

Zusätzlich gibt es eine `Verein`-Klasse, die nur ein Attribut hat:

- `menschen`: Eine Liste von Objekten der Klasse `Mensch`, welche die Mitglieder des Vereins darstellen.

Operator-Überschreibung

1. **Mensch-Klasse**: Wenn man zwei Objekte der Klasse `Mensch` mit dem `+`-Operator verknüpft, wird ein neues Objekt der Klasse `Verein` erstellt. Dieser `Verein` enthält eine Liste, in der die beiden Menschen als Mitglieder hinzugefügt werden.

2. **Verein-Klasse**: Die `Verein`-Klasse hat ebenfalls eine `+`-Operator-Überschreibung. Diese erlaubt es, entweder:

    - Einen neuen `Mensch` zu einem bestehenden Verein hinzuzufügen, oder
    - Zwei Vereine zusammenzuführen, wobei die Mitglieder beider Vereine in einer neuen Liste vereint werden.