V9 Fahrzeug

Mixins

Ein Mixin ist ein Konzept in Dart, das es ermöglicht, Methoden und Eigenschaften von mehreren Klassen wiederzuverwenden, ohne von ihnen abzuleiten. Mixins bieten Flexibilität in der Kombination von Funktionalitäten und fördern die Wiederverwendbarkeit von Code.

Verwendung von Mixins

Um einen Mixin in Dart zu erstellen, definieren Sie eine Klasse, die keine Instanzen erzeugt, und verwenden Sie das Schlüsselwort `with`, um die Mixins in einer anderen Klasse zu implementieren. Hier sind einige wichtige Punkte zur Verwendung von Mixins:

1. **Definition eines Mixins:**
    - Eine Mixin-Klasse wird wie jede andere Klasse definiert, kann jedoch nicht instanziiert werden, weil sie einfach kein Konstruktor hat.
    - Sie können Methoden und Eigenschaften in dieser Klasse definieren.

```
mixin Maschinengewehr {
  int kugelRadius = 10;
  int feuerrate = 20;
  int magazinkapazitaet = 10000;
}
```

2. Implementierung eines Mixins: - Verwenden Sie das Schlüsselwort `with`, um den Mixin in eine andere Klasse zu integrieren.

```
class KriegsMobil with Maschinengewehr{}
```

- Eine Klasse kann mehrere Mixins verwenden, was die Funktionalität erweitert.

```
class KriegsMobil with Maschinengewehr,Rakete{}
```

3. privatisierung eine Mixin In Dart können Mixins durch das `on`-Symbol auf eine bestimmte Basisklasse beschränkt werden. Dies bedeutet, dass der Mixin nur von Klassen verwendet werden kann, die von einer bestimmten Klasse erben oder eine bestimmte Klasse implementieren. Auf diese Weise kann die Wiederverwendung des Mixins auf Klassen mit gemeinsamen Merkmalen oder Funktionalitäten eingeschränkt werden. Zum Beispiel möchte ich das MobilHause-Mixin exklusiv für die Klasse Fahrzeug bereitstellen. Die Klassen Fahrer oder Reifen dürfen das MobilHause-Mixin nicht verwenden.

```
mixin MobilHaus on Fahrzeug {
 double wohnFlache = 30;
 int sitzPlatzAnzahl = 4;
 int schlafPlatzAnzahl = 2;
}
```

Warum Mixins?

Situationserklärung

Stellen wir uns vor, wir haben eine Fahrzeug-Superklasse und deren Unterklassen, die verschiedene Verkehrsmittel repräsentieren. Diese Klassen können jedoch nur als normale Fahrzeuge dienen und sind nicht in der Lage, zusätzliche Funktionalitäten wie Kriegsfahrzeuge oder Wohnfahrzeuge zu unterstützen.

Um diese Fähigkeit hinzuzufügen, könnten wir Superklassen für Kriegsfahrzeuge, Transportfahrzeuge und Wohnfahrzeuge erstellen. Aber bei dieser Lösung entsteht ein Problem:

- Unter Kriegsfahrzeuge könnten wir wiederum verschiedene Verzweigungen haben, wie Fahrzeuge mit Raketen oder Fahrzeuge mit Maschinengewehren.
- Mit jeder neuen Entwicklung würden die Verzweigungen der Fahrzeugklassen immer größer und komplexer werden, was die Lesbarkeit und Erweiterbarkeit des Codes stark beeinträchtigen würde.

Lösung mit Mixins

Hier kommen **Mixins** ins Spiel. Anstatt die Klassenhierarchie immer weiter zu verkomplizieren, verwenden wir Mixins, um gezielt bestimmte Funktionalitäten hinzuzufügen. In unserem Fall erstellen wir die folgenden Mixins:

- **Raketen-Funktionalität**
- **Maschinengewehr-Funktionalität**
- **Wohn-Funktionalität**

Diese Mixins repräsentieren die zusätzlichen Funktionen, die Fahrzeuge je nach Bedarf haben können. Sie fungieren als **Bausteine**, die auf die entsprechenden Klassen angewendet werden, ohne eine tiefe Verzweigung der Klassenstruktur zu erfordern.

Vorteile der Mixins

1. **Bessere Lesbarkeit:**
   Durch die Verwendung von Mixins bleibt der Code übersichtlich, und die Funktionalitäten sind klar voneinander getrennt. Anstatt tief verschachtelte Klassenhierarchien zu erstellen, können spezifische Funktionen modular hinzugefügt werden.

2. **Erweiterbarkeit:**
   Neue Funktionalitäten lassen sich leicht durch neue Mixins hinzufügen, ohne bestehende Klassenhierarchien zu ändern. Man kann neue Mixins für jede neue Funktionalität erstellen und nach Bedarf anwenden.

3. **Flexibilität:**
   Klassen können nach Belieben kombiniert werden. Ein Fahrzeug kann sowohl Raketen als auch Maschinengewehre haben oder nur eine der beiden Funktionen, abhängig von den verwendeten Mixins.

Wichtige Hinweise für Mixins

1. `with` **bei Implementierung:**
   Wenn sowohl `implements` als auch `with` verwendet werden, sollte die Implementierung am Ende hinzugefügt werden.

2. **Reihenfolge bei Vererbung:**
   Im Gegensatz zur Implementierung muss bei der Vererbung die Reihenfolge beachtet werden. Zuerst wird `extends` für die Superklasse verwendet, gefolgt von `with` für die Mixins. Dies bedeutet, dass Mixins nach der Vererbung der Basisklasse hinzugefügt werden.

   Beispiel:

```
class SuperClass {}

class SubClass with Mixin1, Mixin2 implements SuperClass {}

class SubClass2 extends SuperClass with Mixin1, Mixin2 {}
```

3. **Mehrere Mixins und Konflikte:** Eine Klasse kann mehrere Mixins verwenden. In diesem Fall könnte es vorkommen, dass zwei oder mehr Mixins Attribute oder Methoden mit demselben Namen haben. Es wird kein Kompilierungsfehler auftreten, aber die Methode oder das Attribut des zuletzt definierten Mixins überschreibt die anderen. Das kann zu unerwartetem Verhalten führen, wenn nicht vorsichtig benannt wird. **Beispiel:**

   In diesem Fall ist der **gültige** Name Mixin2.

```
mixin Mixin1 {
  String name = 'Mixin1';
}
mixin Mixin2 {
  String name = 'Mixin2';
}

class SubClass with Mixin1, Mixin2 {}
```

Aufgabe

Menschklasse Ein Mensch kann Krankheiten haben. Erstellen Sie ein Krankheit-Mixin mit den **nullable** Attributen: String? krankheitsName, String? gefährlichkeitsNiveau, DateTime? krankheitsAnfangDatum und DateTime? krankheitsEndDatum sowie den Methoden erkranken() und heilen().

Bauen Sie dieses Mixin in die `Mensch`-Klasse ein und testen es, um zu sehen, wie sich die Klasse verändert.