V4 Autoklass

Fehlertoleranz in Klassen

Die Klasse ist der wichtigste Baustein des Codes, da sowohl Daten als auch Widgets nichts anderes als Klassen sind. Daher sollten Klassen mit großer Sorgfalt erstellt werden. Es ist wichtig, falsche Eingaben – sei es durch unbewusste Entwickler, versehentliche Fehler oder fehlerhafte Funktionalitäten – zu erkennen und zu behandeln.

Eine gut gestaltete Klasse sollte in der Lage sein:

1. Falsche Eingaben zu verarbeiten oder zu verhindern.
2. Fehler zu beseitigen oder geeignete Maßnahmen zu ergreifen.
3. Präzise Fehlerwarnungen zu werfen, um das Debuggen zu erleichtern.

Dies verbessert die Fehlertoleranz und stellt sicher, dass der Code robust und zuverlässig ist.

Kapselung (Encapsulation)

Um die Klasse fehlertolerant zu gestalten und wichtige sowie sensible Attribute vor falschen Eingaben zu schützen, wurde das Konzept der **Kapselung (Encapsulation)** entwickelt. Dadurch wird der Zugriff und die Zuweisung bestimmter Attribute eingeschränkt.

Die Kapselung erfolgt durch:

1. **Private Attribute**: Diese Attribute werden vor direktem Zugriff geschützt, indem sie mit einem Unterstrich (_) versehen werden.
2. **Setter- und Getter-Methoden**: Diese Methoden ermöglichen eine kontrollierte Zuweisung und den Zugriff auf die Attribute, wodurch zusätzliche Validierungen durchgeführt werden können, bevor Werte gesetzt oder gelesen werden.

Durch diese Mechanismen wird sichergestellt, dass die Klasse robuster und weniger anfällig für falsche Eingaben ist.

Schritte zur Kapselung (Encapsulation) in Dart

1. **Unterstrich vor dem Attribut**: Um ein Attribut privat zu machen, schreibt man einen Unterstrich (_) vor den Namen des Attributs, z.B. `_name`. Dadurch wird der direkte Zugriff auf das Attribut von außen verhindert.

2. **Private Attribute im Konstruktor nicht direkt verwenden**: Da private Attribute nicht direkt im Konstruktor verwendet werden sollten, definiert man im Konstruktor ein temporäres (mittleres) Attribut, um den übergebenen Wert zu speichern.

3. **Zuweisung mit einem Doppelpunkt (:)**: Nach den runden Klammern des Konstruktors und vor dem Semikolon (`;`) schreibt man einen Doppelpunkt (`:`), gefolgt von der Zuweisung des Werts zum privaten Attribut. Beispiel: `_attribut = attribut;`.

Auf diese Weise wird das gezielte Attribut geschützt, und es kann nach dem Erstellen des Objekts nicht mehr direkt zugewiesen werden. Meistens ist dies jedoch nicht das gewünschte Verhalten. Stattdessen möchte man, dass die Attribute nur vor falschen Eingaben geschützt werden und nicht vollständig unzugänglich sind. Daher verwendet man **Setter** und **Getter**, um die Attribute vor falschen Eingaben zu schützen, während sie dennoch von außen zugänglich und veränderbar bleiben.

Setter

Ein **Setter** ist eine Methode, die es ermöglicht, private Attribute einer Klasse zu ändern, ohne direkt auf sie zuzugreifen. Setter bieten eine kontrollierte Möglichkeit, die Werte zu modifizieren.

In der Regel enthält ein Setter eine Prüfungslogik, um sicherzustellen, dass nur gültige Werte zugewiesen werden. Zum Beispiel kann ein Setter für die maximale Anzahl an Insassen sicherstellen, dass das Auto nicht mehr als 8 und nicht weniger als 1 Insasse haben kann.

Die Verarbeitung von falschen oder ungültigen Eingaben kann auf verschiedene Arten erfolgen. Eine Möglichkeit besteht darin, den gegebenen Wert auf den nächstgelegenen gültigen Wert zu runden oder anzupassen.

```
set maxIinsasseZahl(int value) {
  if (value > 8) {
    _maxInsasseZahl = 8;
    return;
  }
  if (value < 1) {
    _maxInsasseZahl = 1;
    return;
  }
  _maxInsasseZahl = value;
}
```

Oder einfach eine Fehlermeldung geworfen werden, indem eine Exception ausgelöst wird.

```
set maxIinsasseZahl(int value) {
  if (value > 8|| value < 1) {
    _maxInsasseZahl = 8;
    throw Exception('Max IinsasseZahl darf nicht > 8 oder < 1 sein');
  }
   _maxInsasseZahl = value;
}
```

Je nach Situation kann man eine dieser Methoden wählen: entweder den Wert anpassen oder eine Exception werfen.

Getter

- Während der Setter es uns erlaubt, den Wert eines privaten Attributs zu ändern, ermöglicht der **Getter** den Zugriff auf den Wert dieses privaten Attributs.
- In den meisten Fällen wird die Validierung der Attributwerte im Setter durchgeführt, jedoch kann diese bei Bedarf auch im Getter erfolgen.
- Ein weiterer nützlicher Aspekt von Gettern ist, dass sie sich wie separate Attribute verhalten können. Zum Beispiel, wenn wir einen Wert wie das Alter aus einem anderen Attribut, z. B. dem Geburtsdatum, berechnen können, macht es keinen Sinn, beides als separate Attribute zu definieren. Stattdessen könnte das Geburtsdatum als Attribut definiert werden, und das Alter wird mithilfe eines Getters davon abgeleitet. Dadurch verhindern wir fehlerhafte Eingaben, die dazu führen könnten, dass das Alter nicht mit dem Geburtsdatum übereinstimmt.

```
int get alter => DateTime.now().year - _baujahr.year;
```

Konstruktor-Validierung

Falsche Eingaben können auch bei der Definition des Objekts auftreten. In diesem Fall helfen Setter und Getter nicht weiter, daher wird die **Konstruktor-Validierung** verwendet:

1. Anstatt am Ende des Konstruktors ein Semikolon (`;`) zu setzen, verwendet man geschweifte Klammern `{}`.
2. Innerhalb dieser Klammern wird der Validierungsprozess durchgeführt, um sicherzustellen, dass die Eingabewerte korrekt sind, bevor das Objekt erstellt wird.

```
{
    if (maxIinsasseZahl > 8) throw Exception('Max IinsasseZahl darf nicht > 8
sein');
    if (maxIinsasseZahl < 1) throw Exception('Max IinsasseZahl darf nicht < 1
sein');
  }
```

- Es gibt manchmal Attribute, die nicht geändert werden können, sondern nur unter bestimmten Umständen. Zum Beispiel muss die Reifenanzahl eines normalen Autos immer 4 betragen. Im Falle eines Unfalls kann die Anzahl jedoch weniger als 4 sein. Daher darf der Wert nur innerhalb einer Unfall-Methode geändert werden.

```
  void umfall(int verlusteReifen) {
    if(verlusteReifen.isNegative||verlusteReifen>_reifenZahl) return;
    _reifenZahl -= verlusteReifen;
    print('AutoV4 umfällt');
  }
```

Wichtiger Hinweis

Private Attribute sind nur außerhalb der Datei, in der die Klasse definiert ist, geschützt. Innerhalb derselben Datei sind sie weiterhin zugänglich.

Aufgabe

- Geben Sie Ihrer Mensch-Klasse die folgenden Attribute: `nullable` Ehepartner-Vorname und `nullable` Ehepartner-Nachname.
- Schützen Sie Ihre Klasse mithilfe von **Enkapsulierung** (private Attribute, Setter, Getter und Konstruktor-Validierung) vor falschen Eingaben.