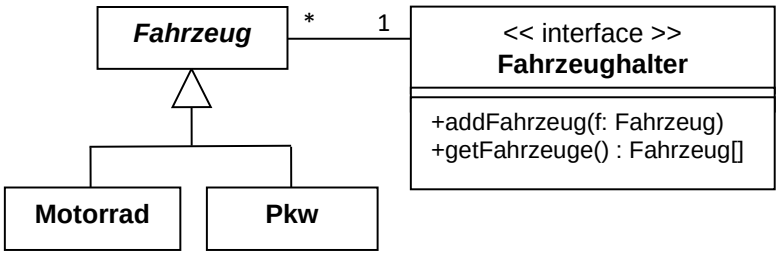
Übungen – Collections

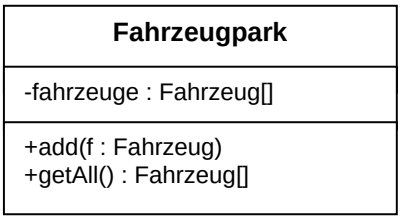
Aufgabe 1 – Mehrere Fahrzeuge

Wir gehen vom bisherigen Stand der Fahrzeughierarchie aus und erlauben nun, zunächst mit Hilfe von Arrays, dass ein Halter mehrere Fahrzeuge haben darf:



Beachten Sie, dass sich die Semantik-Änderung auch im Methodennamen niedergeschlagen hat…

Da das Interface von mehreren Klassen implementiert wird, müsste man nun theoretisch in allen implementierenden Klassen nun das Array-Handling einbauen. Hier hilft man sich gerne mit einer Support-Klasse, die dann von allen implementierenden Klassen als Helferlein benutzt werden kann. Diese könnte wie folgt aussehen:



Die Klasse implementiert bewusst nicht das Interface und hat auch bewusst nicht die Methoden aus dem Interface. Sie dient nur in allen Implementoren von `Fahrzeughalter` als Realisierungshilfe für das mühselige Geschäft mit dem Array.

Effektiv haben Sie hier eine Art spezielle, typisierte eigene Collection geschrieben, die immer dann genutzt werden kann, wenn eine unbekannte Anzahl an Fahrzeugen aufbewahrt werden soll.

Bauen Sie diese Hilfsklasse in die Implementoren von Fahrzeughalter (`NatuerlichePerson` und `Firma`) ein.

Aufgabe 2 – Collections in der Warenbestellung

In der Klasse `Bestellung` werden die `Bestellposition`-Objekte bislang in einem Array gehalten. Ein Array an dieser Stelle limitiert die Anzahl der Bestellpositionen und eine Erweiterung müsste „von Hand“ programmiert werden, was man nur selten macht. Für diesen Fall bietet sich ein Collection-Objekt an. Da die Collections dynamisch sind, kann auch die Situation mit der Exception wieder entfallen.

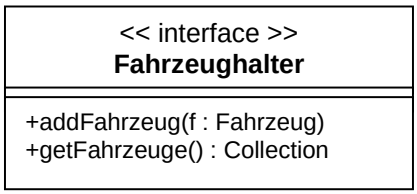
Ersetzen Sie das Array zunächst durch ein `HashSet`-Objekt.

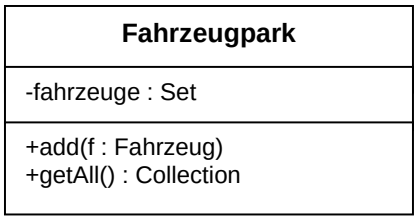
Ersetzen Sie dann das `HashSet`-Objekt durch ein Objekt der Klassen `ArrayList` und `LinkedList`.

Aufgabe 3 – Fahrzeuge mit Collections

Für die Aufnahme der Fahrzeuge in der Klasse `Fahrzeugpark` wurde bisher ein Array genommen. Der Umgang mit Arrays ist etwas umständlich, wenn man eine automatische Kapazitätserweiterung vorsieht. Ersetzen Sie das Array durch ein `Set`.
Dort wo man in den Schnittstellen der Klassen ein Array des Typs `Fahrzeug[]` erhalten hat, soll man nun ein `Collection`-Objekt bekommen.

Interface und Klasse aus dem Diagramm:



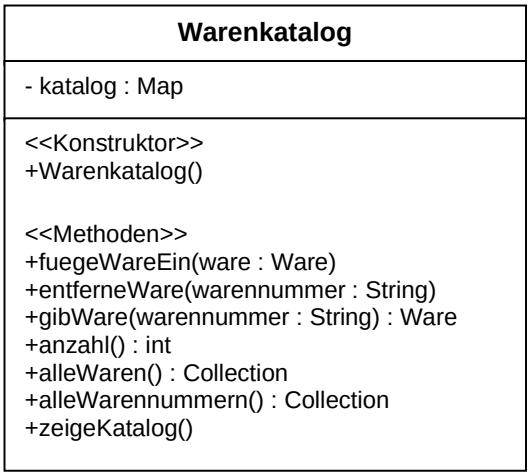


Aufgabe 4 – Ein Warenkatalog

Typischerweise würde man in unserem Bestellsystem die Waren nicht immer frisch anlegen, wenn sie in eine Bestellung aufgenommen werden, sondern die Waren kommen normalerweise aus einer Art Katalogsystem (Datenbank o.ä.). Dafür bieten wir nun eine Zugriffsklasse, den `Warenkatalog`.

(a) Ein Warenkatalog

Wir erstellen einen Warenkatalog, in dem wir eine Serie von Waren einbringen und verwalten können. Die Klasse `Warenkatalog`:



Die Waren sind zunächst in einer `HashMap` abgelegt. Eine `Map` verlangt Key-Value-Paare. Als Key-Objekt ist die Nummer der Ware zu nehmen. Sie bekommt man von der Ware auf Anfrage.

`fuegeWareEin()` benutzt die Warennummer als Schlüssel.
`entferneWare()` entfernt die `Ware` mit der spezifizierten Nummer aus dem Katalog.
`gibWare()` liefert die `Ware` mit der spezifizierten Nummer.
`anzahl()` liefert die Anzahl `Ware`-Objekte, die sich im Katalog befinden.
`alleWaren()` liefert eine `Collection` mit den `Ware`-Objekten, die sich im Katalog befinden.
`alleWarenNummern()` liefert eine `Collection` mit allen Nummern der `Ware`-Objekte im Katalog.
Die Methode `zeigeKatalog()` benützt die Methode `alleWaren()`.

Die Frage ist jetzt: Wann und mit welchen Waren ist ein `Warenkatalog` gefüllt? In einer realitätsnäheren Anwendung würden die Waren wohl in einer Datenbank geführt, und die Klasse `Warenkatalog` ist nur eine Serviceschnittstelle zu dieser Datenbank. Wir werden uns hier so behelfen, dass wir einen `Warenkatalog` unter Verwendung der Methode `fuegeWareEin()` testweise „laden“. Machen Sie dies in einer `main()` einer Klasse `WarenkatalogTest`. Das Testprogramm erstellt einen Katalog und zeigt ihn an.

(b) Sortierungen

Sie sehen, dass die Methode `zeigeKatalog()` die Waren in einer nicht nachvollziehbaren Reihenfolge ausspuckt. Dies liegt an der `HashMap`. Das soll sich ändern.

Sortierung nach Warennummern

Nehmen Sie für den Katalog statt einer `HashMap` eine `TreeMap`.
Da wir als Schlüssel-Objekt in der `TreeMap` mit der Warennummer ein `String`-Objekt verwenden, müssen wir für die Sortierung sonst nichts weiter tun, denn die Klasse `String` implementiert das Interface `Comparable`.
Sie werden feststellen, dass die Ausgabe nun nach der Warennummer sortiert erfolgt.

Sortierung nach Bezeichnung

Wenn wir die Waren nach ihrer Bezeichnung sortiert haben wollen, müssen wir mit einem `Comparator`-Objekt arbeiten.
Implementieren Sie eine Klasse `WareBezeichnungComparator`, die das Interface `Comparator` implementiert.
Geben Sie der Klasse `Warenkatalog` eine neue Methode

```
Collection alleWarenNachBezeichnung()
```

die eine `Collection` liefert, in der die Waren nach ihrer Bezeichnung sortiert auftreten.

In dieser Methode können Sie z.B. mit einem `TreeSet` arbeiten. Bei dessen Erzeugung geben Sie dessen Konstruktor einen `WareBezeichnungComparator` mit. Danach lassen Sie sich von der `TreeMap`, in der sich die Waren des `Warenkatalogs` befinden, alle Waren-Objekte geben. Fügen Sie dann dieses Resultat dem `TreeSet` hinzu. Schließlich wird das `TreeSet` als Resultat zurückgeliefert.
Die neue Methode

```
zeigeKatalogNachBezeichnung()
```

benützt diese Methode.

Sortierung nach Preis

Gehen Sie analog zur Sortierung nach Bezeichnung vor und erlauben Sie auch eine Sortierung nach Preisen.

Aufgabe 5 – JUnit

Experimentieren Sie mit Varianten, wie Sie mittels JUnit zu Testzwecken den Warenkatalog befüllen und testen können.

Hinweis: JUnit kennt z.B. parametrisierte Tests oder man kann Code zur Bestückung von Objekten vor einer Serie von Tests in der Testklasse platzieren…