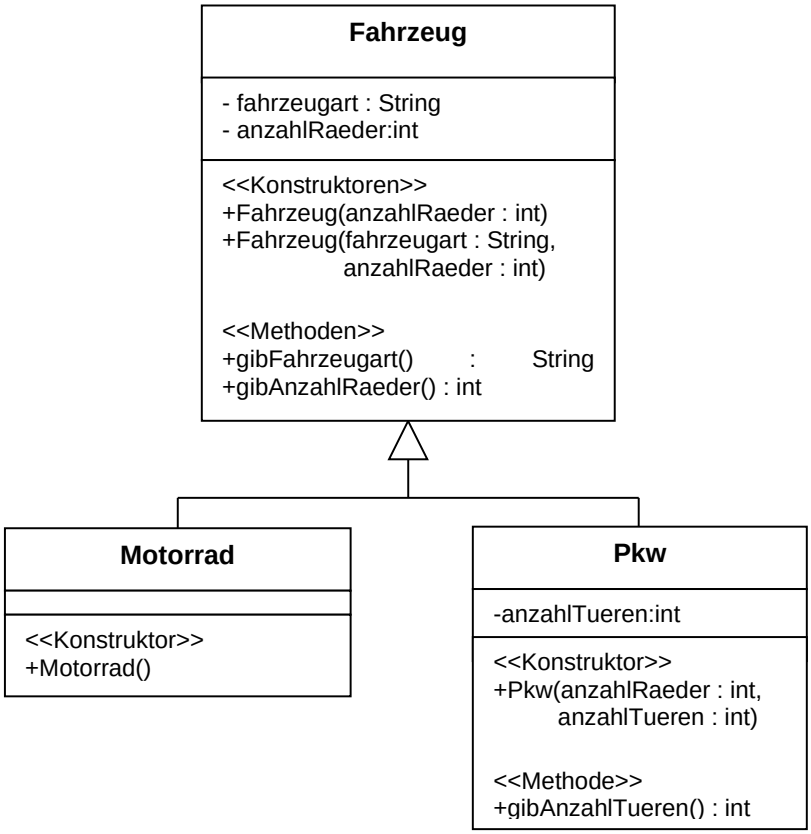
Übungen – Vererbung

Aufgabe 1 – Fahrzeug

Gegeben ist folgendes Klassen-Diagramm:



Implementieren Sie die Klassen nach folgender Vorgehensweise:

1. Die Klasse `Fahrzeug` hat zwei Konstruktoren, einen mit einem Argument und einen mit zwei Argumenten:
   - Der Konstruktor mit einem Argument: Dieser Konstruktor ist für den Fall gedacht, dass ein Objekt dieser Klasse mit `new` erzeugt wird. Er bekommt die Anzahl Räder im Argument übergeben. Die Fahrzeugart wird mit dem weiter unten in der Tabelle angegebenen Default-Text für ein allgemeines Fahrzeug belegt.
   - Der Konstruktor mit zwei Argumenten: Dieser Konstruktor ist für die Klassen gedacht, die von der Klasse `Fahrzeug` abgeleitet werden. Er wird im Konstruktor der abgeleiteten Klassen aufgerufen, um die Fahrzeugart und die Anzahl Räder weiterzugeben.
2. Beim Konstruktor von `Motorrad` können Sie hier im Beispiel davon ausgehen, dass Motorräder immer nur 2 Räder haben.
3. Beachten Sie bei der Erstellung der Konstruktoren von `Pkw` und `Motorrad`, dass Sie die Verantwortlichkeiten der Superklasse `Fahrzeug` mit berücksichtigen.

4. Die Methode `gibFahrzeugart()` wird von den abgeleiteten Klassen nicht überschrieben, gibt jedoch, je nach Typ, den folgenden, im Attribut `fahrzeugart` abgelegten String zurück:

Klasse	Ausgabetext
`Fahrzeug`	`"allgemeines Fahrzeug"`
`Pkw`	`"Pkw"`
`Motorrad`	`"Motorrad"`

Die `main()`-Methode befindet sich in einer separaten Klasse `FahrzeugTest`. Schreiben Sie ein erstes Testprogramm, so dass folgende Ausgabe erfolgt:

```
Das Fahrzeug ist ein allgemeines Fahrzeug mit 16 Rädern.
Das Fahrzeug ist ein Pkw mit 4 Rädern und 5 Türen.
Das Fahrzeug ist ein Motorrad mit 2 Rädern.
```

Aufgabe 2 – Polymorphismus

Polymorphismus auf der Ebene `Fahrzeug`

Führen Sie in der Anwendung die Methode `toString()` ein, die einen String zurückliefert. Der String soll enthalten:

- die Fahrzeugart,
- die Anzahl Räder und
- gegebenenfalls die Anzahl Türen.

Die Klasse Fahrzeug hat eine `toString()`-Methode.

Die Klasse `Motorrad` überschreibt die Methode `toString()` nicht. Sie kommt mit der Implementierung in der Oberklasse aus.

Die Klasse `Pkw` überschreibt die Methode `toString()`. Sie nutzt dabei das Ergebnis der Methode der Oberklasse, so dass sie nur ihren Teil mit den hinzugekommenen Türen ergänzen muss.

Gestalten Sie, falls nötig, Ihr bisheriges Hauptprogramm so um, dass für die Ausgabe überall die Methode `toString()` aufgerufen wird.

Um den Polymorphismus auf der Ebene `Fahrzeug` zu demonstrieren, soll ein Array mit Komponenten des Typs `Fahrzeug` definiert werden. Diese Komponenten werden mit Referenzen auf Objekte verschiedenen, aber kompatiblen Typs belegt. Danach wird an den Referenzen in den Komponenten der Reihe nach (implizit) die Methode `toString()` aufgerufen. Also:

```
Fahrzeug[] fahrzeuge = new Fahrzeug[] {
  new Fahrzeug(16), new Pkw(4, 5), new Motorrad() };

for (Fahrzeug f : fahrzeuge)
{
  System.out.println("Das Fahrzeug ist ein " + f);
}
```

Die Ausgabe:

```
Das Fahrzeug ist ein allgemeines Fahrzeug mit 16 Rädern.
Das Fahrzeug ist ein Pkw mit 4 Rädern. Der Pkw hat 5 Türen.
Das Fahrzeug ist ein Motorrad mit 2 Rädern.
```

Aufgabe 3 – Die Klasse Bruchzahl leicht modifiziert

(a)

In der bisherigen Lösung gibt es die Methode `zeigeAn()`, die eine String-Repräsentation eines `Bruch`-Objekts ausdruckt. Diese Methode soll jetzt etwas umgeschrieben werden in `toString()`, so dass damit die von `Object` geerbte Methode überschrieben wird, d.h. die Methode gibt nur noch die String-Repräsentation zurück und druckt den Bruch nicht mehr selbst aus. Dies bleibt dem Aufrufer überlassen.

Teile der Test-Routine verändern und vereinfachen sich damit zu:

```
r = p.addiere(q);
System.out.println(p + " + " + q + " = " + r);

r = p.subtrahiere(q);
System.out.println(p + " - " + q + " = " + r);

r = p.multipliziere(q);
System.out.println(p + " * " + q + " = " + r);

r = p.dividiere(q);
System.out.println(p + " / " + q + " = " + r);
```

wobei `r` und `q` zwei `Bruchzahl`-Objekte sind.

(b)

Überschreiben Sie in der Klasse `Bruchzahl` die von `Object` geerbte Methode `equals()`, und testen Sie die Methode.

(c)

Passt Ihre Implementierung von `equals` zu den geforderten Kriterien? Testen Sie Ihre Lösung auf alle Varianten. Wenn ja, was müssten Sie tun, damit Sie die einzelnen Kriterien verletzen? Wenn nein – korrigieren Sie Ihre Implementierung, damit die Lösung stimmig ist.