Erste Schritte in BlueJ
Festigung eigene Klasse

Wir wollen jetzt ein Programm schreiben, welches den Benzinverbrauch eines Autos simuliert. Unsere neue Klasse heißt demzufolge Auto.

Schritt 1 - Neue Klasse erzeugen

Erzeugen Sie ein neues leeres Projekt und dann eine Klasse Auto.

Schritt 2 - Quelltext abspecken

Doppelklicken Sie auf die Klasse Auto und entspecken Sie den Quelltext, so wie gelernt. Entfernen Sie

- alle überflüssigen Kommentare und
- die Beispiel-Methode sowie
- alle Zeilen, die sich auf das Beispiel-Attribut x beziehen

Schritt 3 - Attribute definieren

Die Klasse Auto hat noch keine Attribute. Die wollen wir jetzt definieren.
Welche Attribute brauchen wir für eine solche Simulation überhaupt? Auf jeden Fall müssen wir wissen, wie viele Liter Benzin der Tank fasst (z.B. 70 l), dann muss der aktuelle Tankinhalt bekannt sein, mit dem das Auto startet (z.B. 30 l). Wichtig sind natürlich der km-Stand (z.B. 12400) sowie der Verbrauch des Autos (z.B. 9 l / 100 km).
Ergänzen Sie den Quelltext also folgendermaßen:

```
1  public class Auto
2  {
3   // Attribute
4      double tankvolumen, benzinstand;
5      double verbrauch, kmstand;
6
7   // Konstruktoren
8      public Auto()
9      {
10     }
11
12  // Oeffentliche Methoden
13
14  // Private Methoden
15 }
```

1 *Die vier Attribute*

Die vier Attribute setzen wir vorsichtshalber auf den Datentyp double (Komma-Zahlen, das Komma wird als Punkt geschrieben), damit wir später keine Probleme beim Rechnen bekommen. Den Konstruktor lassen wir erst mal leer, und öffentliche und private Methoden gibt es in diesem Stadium der Klasse noch nicht.
Ergänzen Sie das Schlüsselwort "private" vor jedes der vier Attribute.

Schritt 4 - Die Konstruktoren

Als nächstes ergänzen wir den Konstruktor so, dass die Attribute mit sinnvollen Anfangswerten versehen werden. Ein solches Vorgehen bezeichnen wir auch als Initialisierung der Attribute.

```
1  public class Auto
2  {
3   // Attribute
4      double tankvolumen, benzinstand;
5      double verbrauch, kmstand;
6
7   // Konstruktoren
8      public Auto()
9      {
10         tankvolumen =  70.0; // L
11         benzinstand =  50.0; // L
12         verbrauch   =   8.6; // L pro 100 km
13         kmstand     = 77000; // km
14     }
15
16  // Oeffentliche Methoden
17
18  // Private Methoden
19 }
```

2 *Die Initialisierung der Attribute*

Es gibt zwei Möglichkeiten, ein Attribut zu initialisieren, die Abb. 2 zeigt die erste Möglichkeit:
Erstens: Direkte Wertzuweisung an das Attribut. Das Tankvolumen wird hier zum Beispiel auf den Wert 70.0 gesetzt, und der Benzinverbrauch auf den Wert 8.6. Der Benutzer des fertigen Java-Programms hat keine Möglichkeit, hier andere Zahlen einzugeben, es sei denn, er schreibt eigene manipulierende Methoden hierfür.
Zweitens: Der Konstruktor kann wie alle anderen Methoden auch Parameter einsetzen, um Informationen von außen zu holen.

```
1  public class Auto
2  {
3   // Attribute
4      double tankvolumen, benzinstand;
5      double verbrauch, kmstand;
6
7   // Konstruktoren
8      public Auto()
9      {
10         tankvolumen =  70.0; // L
11         benzinstand =  50.0; // L
12         verbrauch   =   8.6; // L pro 100 km
13         kmstand     = 77000; // km
14     }
15
16     public Auto(double tv, double bst, double verb, double km)
17     {
18         tankvolumen =  tv;
19         benzinstand =  bst;
20         verbrauch   =  verb;
21         kmstand     =  km;
22     }
23
24  // Oeffentliche Methoden
25
26  // Private Methoden
27 }
```

3 *Ein zweiter Konstruktor mit Parametern*

In Abb. 3 haben wir einen zweiten Konstruktor mit vier Parametern in die Klasse eingebaut. Hier kann der Benutzer der Klasse beim Erzeugen eines Auto-Objektes selbst die Startwerte der vier Attribute bestimmen, indem er dem Konstruktor vier Werte übergibt.
Außerdem sehen wir hier, dass eine Klasse zwei, drei oder noch mehrere Konstruktoren haben kann. Allerdings müssen sich die Konstruktoren in ihren Parametern eindeutig voneinander unterscheiden. Man nennt diese technische Möglichkeit auch "overloading" von Methoden. Auch bei normalen sondierenden oder manipulierenden Methoden ist ein solches overloading erlaubt.
Initialisierung:
Die Deklaration einer Variablen reicht meistens nicht aus. Oft muss eine Variable auch noch initialisiert werden. Das heißt, man weist ihr einen Anfangswert zu.

```
24  // Oeffentliche Methoden
25      public void tanken(double liter)
26      {
27          benzinstand = benzinstand + liter;
28      }
29
30      public void anzeigen()
31      {
32          System.out.println("Tankvolumen = "+tankvolumen+" Liter");
33          System.out.println("Benzinstand = "+benzinstand+" Liter");
34          System.out.println("Verbrauch   = "+verbrauch  +" Liter/100 km");
35          System.out.println("km-Stand    = "+kmstand    +" km");
36      }
37
38  // Private Methoden
39  }
```

4 *Die neuen Methoden tanken und anzeigen*

1. Übung
Ergänzen Sie die Klasse Auto um die in Abb. 4 gezeigten Methoden tanken() und anzeigen(). Die Methode tanken() ist noch fehlerhaft.
a) Warum ist die Methode noch fehlerhaft; was könnte man besser machen?
b) Verbessern Sie die Methode entsprechend!

2. Übung
Folgende Methode fahren() simuliert das eigentliche Autofahren:

```
40      public void fahren(double kilometer)
41      {
42          kmstand = kmstand + kilometer;
43          benzinstand = benzinstand - kilometer/100 * verbrauch;
44      }
```

5 *Die erste Fassung der Methode fahren*

Auch diese Methode funktioniert noch nicht fehlerfrei.
a) Wo liegt der Fehler (was kann theoretisch passieren?)
b) Verbessern Sie diesen Fehler (und alle anderen, die Sie finden)

3. Übung
Autofahren ist weitaus komplizierter, als die vorangegangenen Übungen es ahnen lassen. So hängt z.B. der Verbrauch von dem Gesamtgewicht des Fahrzeugs ab. Sitzt nur der Fahrer im Auto, so wird weniger Benzin verbraucht, als wenn er drei schwere Fahrgäste transportiert und auch noch deren Gepäck.

Ergänzen Sie die Klasse Auto um die nötigen Attribute und Methoden, um ein solches Verhalten zu realisieren!
Hinweise:
Jedes Auto hat ein Leergewicht sowie ein zulässiges Gesamtgewicht. Außerdem können maximal vier Personen zusteigen. Jede Person wiederum hat ein bestimmtes Gewicht. Außerdem muss das Zuladen von Gepäck möglich sein. Für den Benzinverbrauch muss eine Formel entwickelt werden, bei der der Verbrauch pro 100 km mit dem Gesamtgewicht steigt.

5. Übung

Das Thema Autofahren ist derart komplex und interessant, dass Sie sicherlich auch noch eine eigene Idee haben, wie man die Klasse Auto realistisch erweitern könnte. Da auch die eigene Kreativität in der Informatik eine Rolle spielen sollte, gibt es diese offene Aufgabe.
Ergänzen Sie die Klasse Auto um Attribute und Methoden, die das Fahren noch realistischer machen!