

## BlueJ - Teil 4

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

### Schritt 1 - Neue Klasse erzeugen

Erzeugt - so wie in Teil 2 gelernt - eine neues leeres Projekt und dann eine Klasse Auto.

### Schritt 2 - Quelltext abspecken

Doppelklickt auf die Klasse Auto und entspeckt den Quelltext (falls nötig), so wie in Teil 3 gelernt.

Entfernt

- 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.

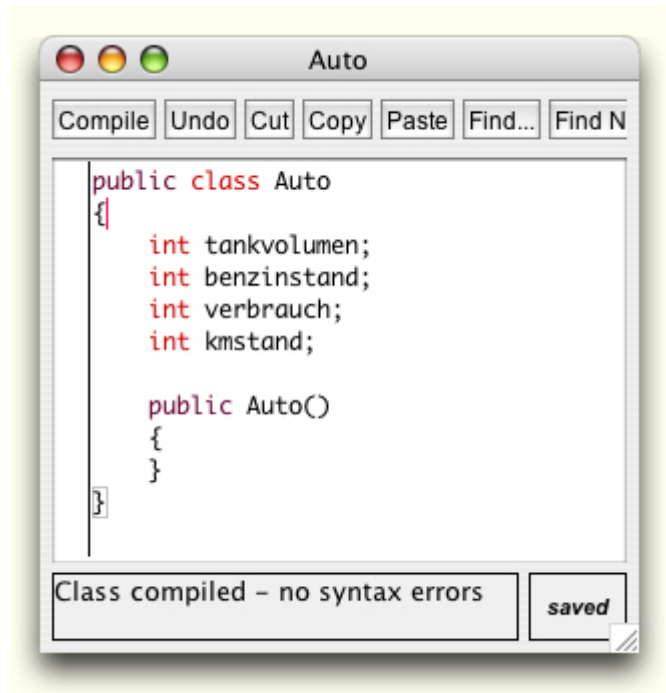
Welche Attribute brauchen wir für eine solche Simulation überhaupt?

- 1.
- 2.
- 3.
- 4,
- 5.
- 6.

Von was hängt die Auswahl der Attribute überhaupt ab?

Für diese Simulation brauchen wir nur ganz bestimmte Attribute.

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 ist natürlich auch der km-Stand (z.B. 12.400) sowie der Verbrauch des Autos (z.B. 9 l / 100 km). Ergänzt den Quelltext also folgendermaßen:



```
public class Auto
{
    int tankvolumen;
    int benzinstand;
    int verbrauch;
    int kmstand;

    public Auto()
    {
    }
}
```

Hinweis 1:

In diesem Beispiel arbeiten wir mit int-Zahlen. Welche Variablenform würde sich ebenfalls eignen, bzw. die Berechnungen sogar verbessern?

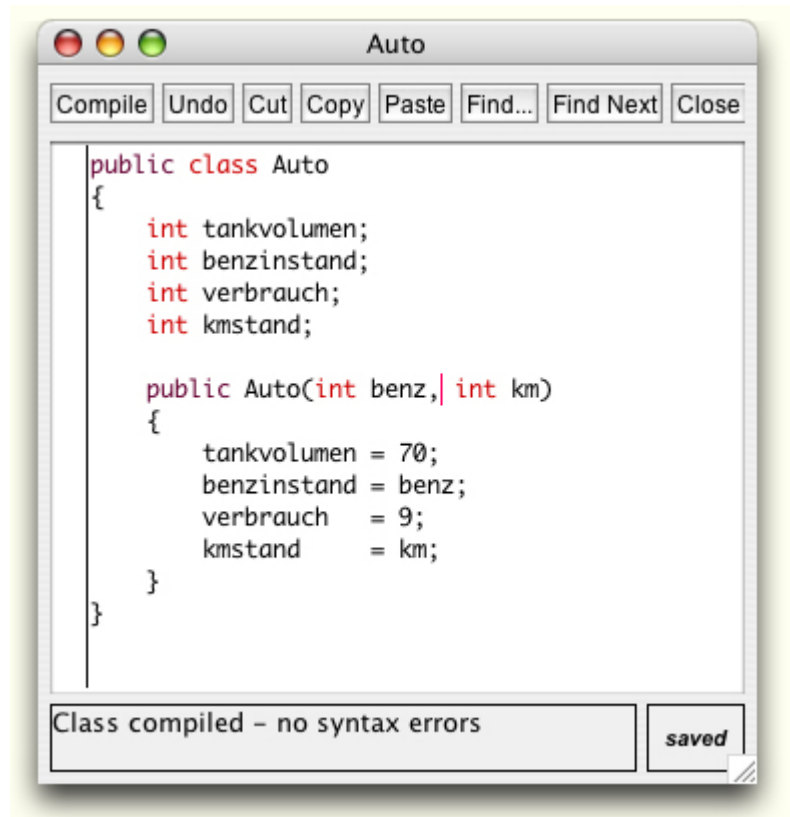
Hinweis 2:

Wenn Ihr wollt, könnt ihr das Schlüsselwort "private" vor jedes der vier Attribute setzen. Das würde noch stärker dem Prinzip der **Datenkapselung** entsprechen.

*Datenkapselung? Wasn das? Googelt mal!*

## Schritt 4 - Die erste richtige Methode

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.



```
public class Auto
{
    int tankvolumen;
    int benzinstand;
    int verbrauch;
    int kmstand;

    public Auto(int benz, int km)
    {
        tankvolumen = 70;
        benzinstand = benz;
        verbrauch = 9;
        kmstand = km;
    }
}
```

Es gibt zwei Möglichkeiten, ein Attribut zu initialisieren, die Abb. 2 zeigt beide Möglichkeiten:

Erstens: Direkte Wertzuweisung an das Attribut. Das Tankvolumen wird hier zum Beispiel auf den Wert 70 gesetzt, und der Benzinverbrauch auf den Wert 9. Der Benutzer des fertigen Java-Programms hat keine Möglichkeit, hier andere Zahlen einzugeben.

Auf den Alltag übertragen könnte man das so interpretieren: Der Kunde hat sich ein bestimmtes Auto gekauft, und dies hat nun mal einen Benzintank mit einem bestimmten Fassungsvermögen, das der Kunde nicht ändern kann. Und den Verbrauch kann der Kunde normalerweise auch nicht ändern.

Zweitens: Der Konstruktor kann wie alle anderen Methoden auch Parameter einsetzen, um Informationen von außen zu holen. Oben hat der Konstruktor zwei solcher Parameter, „benz“ und „km“. Beim Erzeugen eines neuen Objektes vom Typ Auto kann der Benutzer dem Konstruktor jetzt zwei Zahlen mitgeben, die dann als Benzinstand und Kilometerstand interpretiert werden. Die übergebenen Parameter werden im Konstruktor den Attributen zugewiesen.

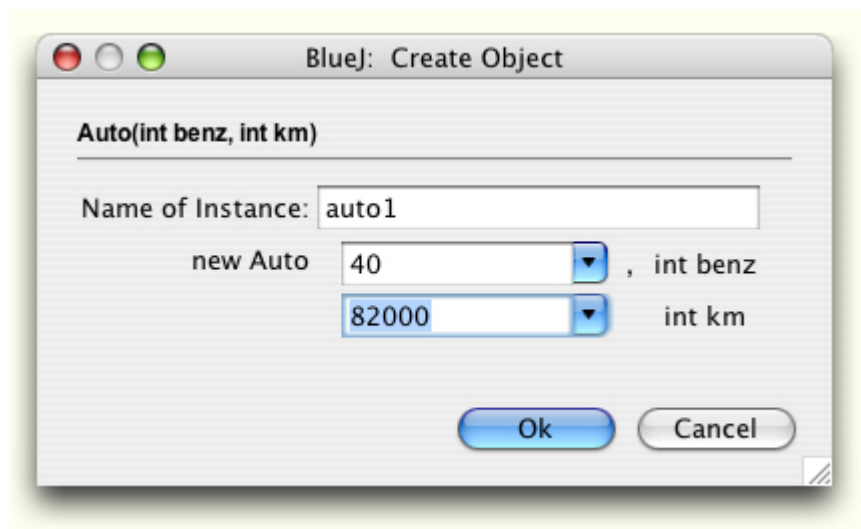
Aufgeschrieben:

*Initialisierung:*

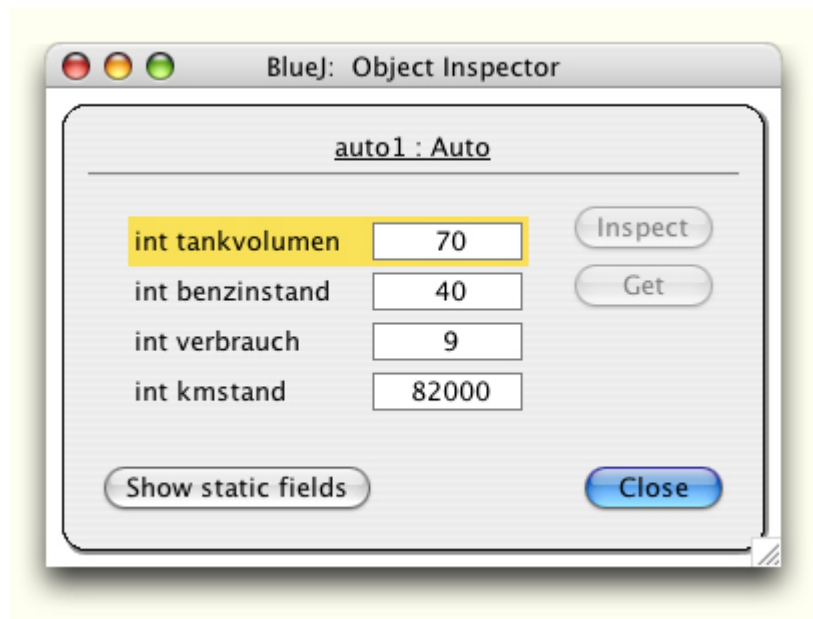
*Die Deklaration einer Variable reicht meistens nicht aus. Oft muss eine Variable auch noch initialisiert werden. Das heißt, man weist ihr einen Anfangswert zu.*

## Schritt 5

Kompiliert den Quelltext und erzeugt dann ein Objekt der Klasse Auto. Dabei werdet ihr von BlueJ nach den beiden Parameterwerten gefragt:



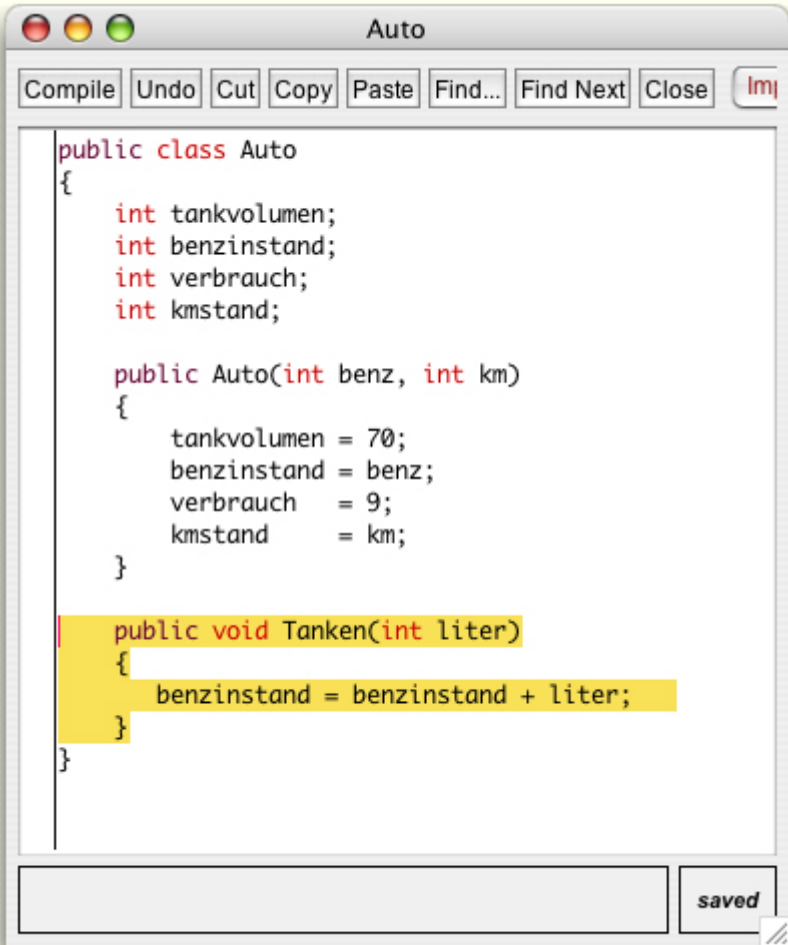
Anschließend klickt wieder mit der rechten Maustaste auf das Objekt auto1 und führt den Befehl „inspect“ aus. Er müsste jetzt folgendes Fenster erscheinen, welches den Status des Objektes auto1 anzeigt:



## TIPP

*Macht reichlich Gebrauch von dem ObjektInspektor! Mit diesem mächtigen Instrument könnt ihr die Werte von Attributen überwachen und dadurch Programmierfehlern auf die Schliche kommen.*

## Schritt 6 - Übung 4.1



```
public class Auto
{
    int tankvolumen;
    int benzinstand;
    int verbrauch;
    int kmstand;

    public Auto(int benz, int km)
    {
        tankvolumen = 70;
        benzinstand = benz;
        verbrauch = 9;
        kmstand = km;
    }

    public void Tanken(int liter)
    {
        benzinstand = benzinstand + liter;
    }
}
```

Ergänzt die Klasse Auto um die oben gezeigte Methode tanken(). Führt sie aus. Was passiert, wenn ihr 90 Liter Benzin tankt, der Tank aber bereits 40 Liter enthält und maximal 70 Liter fasst? Verbessert die Methode so, dass das Tankvolumen (also der maximale Tankinhalt) nicht überschritten werden kann!

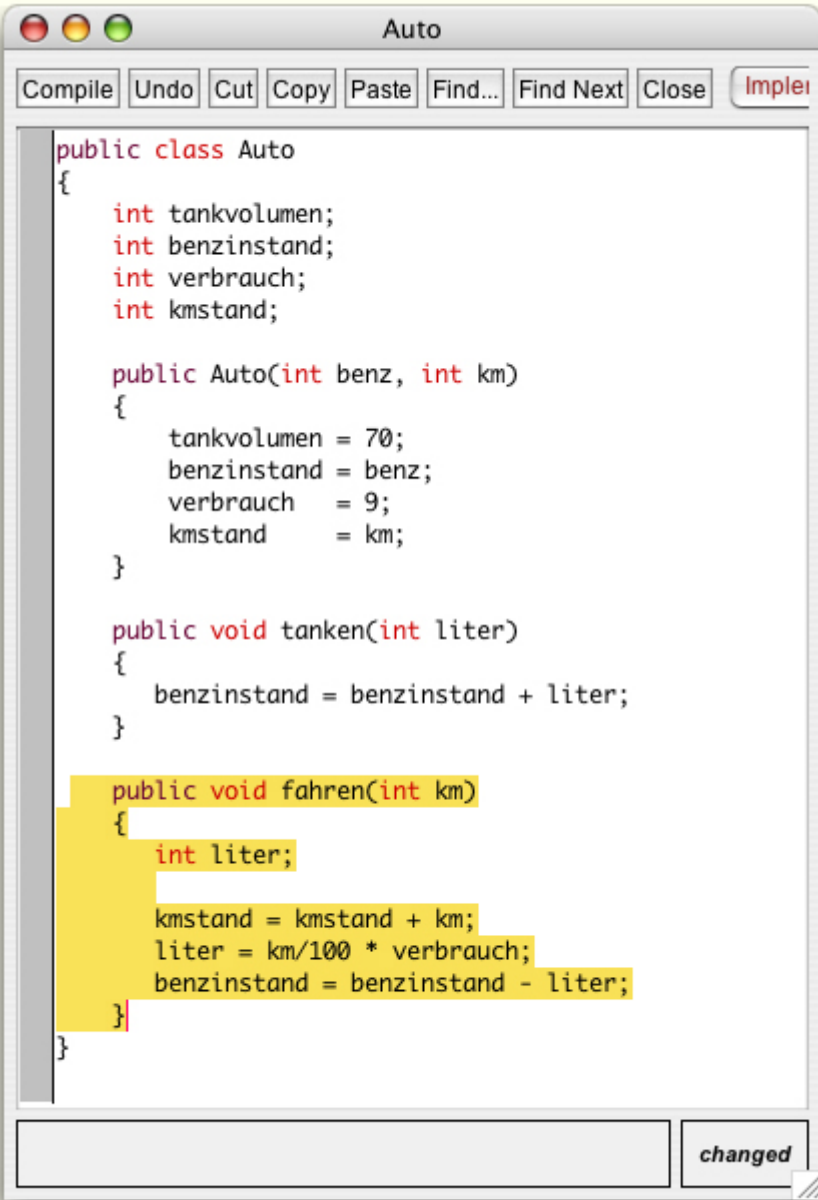
Design, bzw. Visualisierung: wenn ihr das Auffüllen des Benzintanks irgendwie anschaulich simuliert, und zwar so, dass man sieht, wie der Tank langsam voller wird, seid ihr die Helden der Programmierung!

Wie wäre sowas zu veranschaulichen?

Also eine for- oder while-Schleife solltet ihr dann schon einsetzen, eventuell könnt ihr den Benzinstand mit Hilfe von Sternchen oder anderen Zeichen symbolisieren.

## Schritt 7 - Übung 4.2

Folgende Methode fahren(int km) simuliert das eigentliche Autofahren:



```
public class Auto
{
    int tankvolumen;
    int benzinstand;
    int verbrauch;
    int kmstand;

    public Auto(int benz, int km)
    {
        tankvolumen = 70;
        benzinstand = benz;
        verbrauch = 9;
        kmstand = km;
    }

    public void tanken(int liter)
    {
        benzinstand = benzinstand + liter;
    }

    public void fahren(int km)
    {
        int liter;
        kmstand = kmstand + km;
        liter = km/100 * verbrauch;
        benzinstand = benzinstand - liter;
    }
}
```

Auch diese Methode funktioniert noch nicht fehlerfrei. Sie können zum Beispiel 1000 km fahren und dabei 90 Liter Benzin verbrauchen, obwohl Sie nur 40 Liter im Tank hatten.

Verbessert die Methode, so dass das nicht passieren kann.

Fertig? Bevor SchülerVZ geöffnet wird, wäre es klasse, wenn ihr auch das Autofahren anschaulich simuliert, so ähnlich, wie ihr das bereits beim Tanken gemacht habt.