1.1 Einfache Theorie der Objektorientierten Programmierung

Zunächst klären wir am Beispiel Vogel, was eine **Klasse** ist: Ähnlich wie in der Biologie gruppiert man in der OOP (Objektorientierten Programmierung) Objekte und fasst sie in Klassen zusammen. Dabei stellt sich die Frage, was einen Vogel auszeichnet:

Ein Vogel hat bestimmte **Eigenschaften (oder Attribute)**. Er hat zum Beispiel eine bestimmte Farbe, ein Geschlecht und man kann seine Flügelspannweite messen.

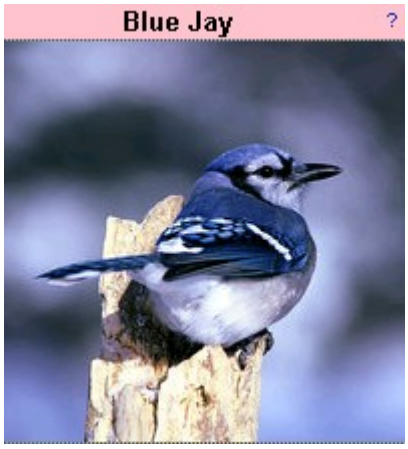
Ein Vogel kann bestimmte Dinge tun. Er kann etwa *singen* oder ein Ei legen. Das sind die so genannten **Methoden (Operationen)** des Vogels (singe(...), legeEi(...)).

Methoden teilt man in zwei Gruppen ein:

- Es gibt zum einen b**eobachtende** oder auch **sondierende Methoden**. Diese beantworten z.B. die Frage: Welche Farbe hast du? Im Programm werden sie *gibFarbe*() oder *getFarbe*() genannt.
- Zum anderen gibt es Methoden, die Eigenschaften ändern können (**verändernde Methode**). Ein Beispiel wäre *setFarbe("gruen")*.

Aus der Klasse Vogel lassen sich konkrete **Objekte** bilden: z.B. vogel1, vogel2, vogel3. Jedes Objekt hat einen bestimmten **Zustand**. So kann z.B. vogel1 die Farbe gelb haben, weiblich sein und 40 cm Spannweite haben. der Zustand von vogel2 könnte sein: gelb, 30 cm Spannweite, männlich. **Während die Klasse Vogel etwas abstraktes ist (eine Art Bauplan für Objekte) ist ein Objekt konkret.** Was das genau bedeitet, erfahren Sie, wenn Sie die folgenden Beispiele nachvollziehen.

Abbildung 1: BlueJ Logo
Blue Jay Vogel: http://en.wikipedia.org/wiki/Blue_Jay



Zusammenfassung

- **Objekte** sind in der objektorientierten Programmierung Daten (**Eigenschaften** oder auch **Atrribute**) und die damit verknüpfte Programmlogik (**Methoden** oder auch **Operationen**), die zu Einheiten, nämlich eben den Objekten, zusammengefasst sind.
- Gleichartige Objekte werden zu **Klassen** zusammengefasst.
- **Klassen dienen als Vorlage (wie ein Bauplan)** zur Herstellung von Objekten. Von einer Klasse können beliebig viele Objekte hergestellt werden. Die **Objekte sind einzigartig**, da sie einen unterschiedlichen Namen tragen müssen, obwohl ihr Zustand identisch sein kann.
- Der **Zustand** (oder auch **Status**) ist die Gesamtheit der Werte der Eigenschaften.

nach http://de.wikipedia.org/wiki/Objekt_%28objektorientierte_Programmierung%29
und nach: http://www.u-helmich.de/inf/BlueJ/kurs11/seite01/theorie.html

1.2 Was ist eine Klasse, was ist ein Objekt?

Die Stempel Analogie

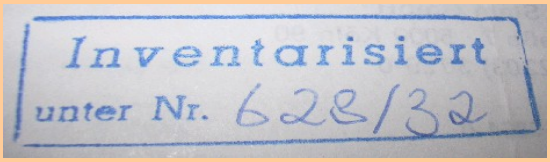


Abbildung 2: Diese Klasse Stempel hätte die Eigenschaft inventarnummer. Der Zustand des Objektes stempel1 ist so, dass die Inventarnummer 628/32 ist.

Sicherlich kennen Sie die Stempel, die in einem von der Schule geliehenes Schulbuch abgedruckt sind: Dort ist eine Tabelle vorgegeben, wo Sie das Schuljahr und Ihren Namen eintragen. Das können Sie mit einer Klasse vergleichen: Der Stempel ist die Vorlage also die Klasse, die immer gleiche Stempelabdrücke herstellt, die Objekte. Füllen Sie die Stempelabdrücke aus, so können Sie dies damit vergleichen, dass Sie den Zustand des Stempelabdruckes ändern. Jeder Stempelabdruck ist einzigartig, da Sie keine zwei Stempel genau übereinander anbringen können. Idee: http://www.u-helmich.de/inf/BlueJ/kurs11/seite01/theorie.html

1.2.1 Klassendiagramm und Objektdiagramm

Im **Klassendiagramm** (auch UML-Klassendiagramm genannt) hält man die Elemente einer Klasse anschaulich fest: Klassenname, Eigenschaften und Methoden.

- Im **Klassendiagramm** werden Klassenname, Eigenschaften und Methoden festgehalten.
- Alle **Methoden**, die wir schreiben, sind öffentlich, also **public**, alle **Eigenschaften private** Im Klassendiagramm bekommen Eigenschaften daher ein vorgestelltes – und Methoden ein +.

Übung:

1) Was sind Eigenschaften der Klasse Kartenspiel (Skatspiel)? Was sind die Methoden? Legen Sie ein Klassendiagramm an.
2) Nennen Sie Eigenschaften und Methoden des Objektes Kaffeemaschine. Legen Sie ein Klassendiagramm an.

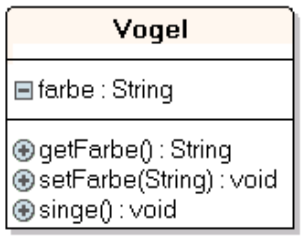


Abbildung 3: Klassendiagramm Vogel

1.3 Java mit BlueJ - Beispiel Onlinebank

Unter dem **Quelltext** oder auch Quellcode (engl. source code) oder Programmcode versteht man in der Informatik den für Menschen lesbaren in einer Programmiersprache geschriebenen Text eines Computerprogrammes. Programme werden in Java als Quelltext geschrieben[1].

1.3.1 Eigenschaften der Klasse Konto

Eine Eigenschaft (Attribut) ist ein strukturelles Merkmal einer Klasse.

- Es besitzt einen Namen und einen **(Daten-)Typen**.
- Konvention: **Eigenschaftsnamen werden klein geschrieben, Klassennamen groß**.

Java kennt verschiedene **Datentypen**. Datentypen legen fest, in welcher Art die Daten sind, die gespeichert werden. Wir beschränken uns zunächst auf vier Datentypen. Welche das sind, sehen Sie in der folgenden Tabelle:

1 Es gibt auch Programmiersprachen, die sich z.B. durch Klicken von Grafiken bedienen lassen. Der Text steht dann nicht im Vordergrund (z.B. Squeak für Grundschulkinder oder die Sprache der Lego Roboter)

Wichtige Datentypen	**Anwendungsbereiche**
String speichert Zeichenketten.	**Zeichenketten:** Namen, Telefonnummern, "Hallo", Chat, Hausnummern
double speichert Zahlen im Bereich von +/-1,7E+308 (also 17 mit 307 Nullen)**.**	**Dezimalzahlen:** Geldbeträge, Wertetabelle, 1.734, -23342323, 0.0021, 1.414
int (sprich: Integer - Ganzzahl) -2147483648 bis 2147483647	**Ganze Zahlen:** -15, 2337356, 3453, 0, Jahreszahlen, Anzahl von Kursteilnehmern, Anzahl von Durchläufen
boolean ist ein Wahrheitswert. Er kann true oder false sein.	Abfragen wie: Wollen Sie wirklich abbrechen? Geschlecht, Logische Aussagen und Abfragen (Ist a größer als 15?)

Übung:
1) Nennen Sie Anwendungen, bei denen der Wertebereich von double bzw. int nicht genügt.
2) Warum sollten Haus- und Telefonnummern als String und nicht als int gespeichert werden?
3) Warum unterscheidet Java double und int? double enthält doch alle int-Zahlen?

1.3.2 Der erste Quelltext der Klasse Konto

```
public class Konto {
  private String besitzerName;
  private double kontostand;
}
```

Nach diesen einführenden Überlegungen geht es nun daran, den **Quelltext** zu erarbeiten. Bei jedem Quelltext muss man sich an die **Syntax** der Programmiersprache richten. In Java beginnen wir damit, eine Klasse zu definieren. Dazu verwenden wir das Wort ***class***.

1.3.3 Was bedeutet public und private?

public und **private** regeln die Zugriffsrechte (Fachbegriff: **Zugriffsmodifizierer**).
Vergleichbar ist das mit dem Zugriff auf eine **Waschmaschine:**

- **Die Bedienknöpfe sind public, also öffentlich und von jedem bedienbar.**
- **Das Innenleben ist private**.

Hier soll keine Hausfrau und erst Recht kein Hausmann Zugriff haben, da sie bzw. er etwas kaputtmachen könnte. Als erste Faustregel genügt es, sich zu merken, dass Attribute private (Innenleben der Klasse) und Klassen public sind (Jeder soll Klassen ausführen dürfen). Im Klassendiagramm zu Vogel haben wir bereits Zugriffsmodifizierer gesehen (siehe Konvention: Eigenschaften immer private, Klassen immer public).

Abbildung 4: Zugriffsrechte Waschmaschine: Der Programmschalter ist public, das Innenleben (Elektronik etc.) private

1.3.4 Schritte zur Eingabe des Quelltextes

Schritte	Wirkung
1) Projekt neu	Legt ein neues Projekt an. Nennen Sie es „Onlinebank“
2) Neue Klasse	Legt eine neue Klasse an. Als Namen wählen Sie „Konto“ (groß geschrieben, da es eine Klasse ist).
3) Rechte Maustaste auf die Klasse Konto (d.h. auf das beige Rechteck) – Bearbeiten	Öffnet den Quelltext. Löschen Sie alles und tippen Sie dann den Quelltext ab. Achten Sie auf Klein- und Groß-schreibung.
4) Drücken Sie auf übersetzen	Die Klasse wird übersetzt. Achten Sie auf die unterste Zeile. Dort werden Fehler angezeigt.
5) Rechte Maustaste auf auf die Klasse Konto „new Konto()“	Legt ein Objekt Konto an. Sie können die Benennung so lassen wie BlueJ es vorschlägt: konto1
6) Doppelklick auf das Objekt konto1 (rotes Rechteck unten)	Öffnet den Objektinspektor.

1.3.5 Der Objektinspektor

Ein Rechtsklick auf das rote Rechteck und dann auf Inspizieren öffnet bei BlueJ den **Objektinspektor** (Abb. 5). Dieser gibt einen Überblick über den **Zustand** des Objektes: Unser Besitzer heißt **null**, ein Wort, das Java verwendet, wenn String-Variablen noch keinen Wert haben. Der Kontostand ist 0, was der Anfangswert für alle numerischen Variablen ist.

Abbildung 5: Der Objektinspektor

1.4 Zusammenfassung

- Eine **Klasse** (be-)schreibt man in BlueJ mit Hilfe eines **Quelltextes**. Der grundsätzliche minimale Aufbau ist:

```
public class Klassenname {
  // Eigenschaften
  private Datentyp eigenschaft1;
  private Datentyp eigenschaft2;
}
```

- Die Zugriffsmodifizierer **public** und **private** regeln die Zugriffsrechte.
  Als **vorläufige Regel** ist die **Klasse immer public** und die **Eigenschaften sind immer private**.
- Aus Klassen lassen sich Objekte erzeugen, die einen **eindeutigen Namen** tragen müssen.
- Die Objekte haben einen Zustand, den man mit Hilfe des **Objektinspektors** ansehen kann. Anfangszustand von Strings ist das englische Wort **null** und von Zahlentypen **0**.
- Ein „Doppelslash“ // am Anfang der Zeile macht aus der Zeile einen Kommentar, den der Computer ignoriert. Ein Kommentar erläutert den Quelltext.

Übung:	
1)	Erweitern Sie die Klasse Konto um die Eigenschaften *kreditlimit* und *telefonnummer*.

Übung:	
	Welche Datentypen würden Sie verwenden?
2)	Entwerfen Sie das Klassendiagramm zur Klasse *Konto*. Entwerfen Sie den Quelltext zur Klasse Vogel.
3)	Profis: Recherchieren Sie zum Thema "Genauigkeit von Fließkommazahlen", nach den Grenzen der Genauigkeit des Typs double. Welche weiteren numerischen Datentypen bietet Java?

1.5 Konstruktor – Wie sind die Ausgangswerte?

Der **Konstruktor** ist eine spezielle Methode die bei der Erzeugung von Variablen aufgerufen wird. Er wird dazu **verwendet, die Variable in einen definierten Anfangszustand** zu versetzen. Dieser Vorgang nennt sich **auch Initialisierung**.
Nach: http://de.wikipedia.org/wiki/Konstruktor

Wir erweitern die Klasse Konto wie folgt:

```
100 public class Konto {
      // (Instanzvariablen der) Eigenschaften
      private String besitzerName;
      private double kontostand;

105   //Konstruktor
      public Konto (String pBesitzerName){
        besitzerName = pBesitzerName;
        kontostand = 1.0;
      }
110 }
```

Begriff	Erläuterung
Konstruktor	legt die Anfangswerte fest. Er erhält den gleichen Namen wie die Klasse selbst: z.B. public Konto Ein Konstruktor ist eine spezielle Methode. Alle werte, die nicht im Kosntruktor festgelegt werden, bekommen einen „default-Wert“. Bei Zahlentypen ist das 0 bei Strings „null“
(Übergabe-)parameter	Sind die Werte, die der Methode übergeben werden. Für jeden Parameter muss ein Datentyp festgelegt werden. Der Name des Parameters kann zur besseren Übersicht mit p beginnen: z.B. pBesitzerName
null	Ist ein leeres Objekt. Es zeigt an, wenn eine Variable bzw. ein Objekt nicht mit einem Wert belegt wurde.
geschweifte Klammern { }	Tippt man mit Strg + Alt + 7 bzw. 0. Sie werden eingesetzt, um das Programm zu strukturieren. Die gesamte Klasse steht in geschweiften Klammern und auch Methoden wie der Konstruktor. Das Weglassen von Klammern ist eine typische Fehlerquelle. Das letzte Zeichen einer Klasse ist „}“ (vgl. oben)
amerikanische Notation	Java ist amerikanisch, daher Punkt statt Komma: 3.141
Semikolon	Nach (nahezu) jedem Befehl möchte Java ein Semikolon ;

1.5.1 Mehrere Parameter

Möchte ich mehrere Parameter übergeben, so trenne ich sie mit einem Komma.Als Beispiel füge ich den Straßennamen hinzu (*Hinweis: Im Quelltext meide ich deutsche Umlaute als Variablennamen*).

```
115 public class Konto {
      // Eigenschaften
      private String besitzerName;
      private String strassenName;
      private double kontostand;
120
      //Konstruktor
      public Konto (String pBesitzerName, String pStrassenName){
        besitzerName = pBesitzerName;
        strassenName = pStrassenName;
125     kontostand = 1.0;
      }
    }
```

Übung:	
1)	Vollziehen Sie das angegebene Beispiel nach. Testen Sie Fehleingaben bei BlueJ: Lassen Sie beim Besitzernamen die Anführungstsriche weg, geben Sie eine Zahl, geben Sie nichts ein...
2)	Lassen Sie statt des Namens den Kontostand als Parameter übergeben.
3)	Ergänzen Sie die Eigenschaften *vorname, telefonnummer, postleitzahl, geburtsjahr, geschlecht und hausnummer*. Begründen Sie dabei, welche Datentypen Sie verwenden.

130 1.6 Methoden – Was kann meine Klasse?

Die Methode einzahlen() als Beispiel für eine verändernde Methode
Es sind natürlich noch viele weitere Methoden möglich. Ein weiteres Beispiel ist die Methode einzahlen().

```
    public void einzahlen(double pBtrag)
135 {
       kontostand += pBetrag;
       // oder kontostand = kontostand + pBetrag;
    }
```

Diese Methode sorgt dafür, dass zu dem bisherigen Kontostand ein bestimmter betrag hinzugefügt
140 wird. Hervorzuheben ist, dass die folgende Zeile nicht mathematisch zu verstehen ist:
kontostand = kontostand + pBetrag;
Das bedeutet: kontostand wird die Summe aus dem *bisherigen* Kontostand und einem Parameter pBetrag **zugewiesen**.

Übung:	
1)	Bauen Sie die Methode einzahlen in Ihre Kontoklasse ein. Testen Sie das Einzahlen von negativen Beträgen, großen Beträgen, 0, Wörtern, Rechnungen...
2)	Entwerfen Sie auf Grundlage von einzahlen die Methode auszahlen().
3)	Entwerfen Sie raubeAus(), die den Kontostand auf 0 setzt.

Zuweisen von Werten bei Variablen

- **Das Gleichheitszeichen wird** in Java **als Zuweisungsoperator verwendet.**
- **Zum Erhöhen einer Variable lässt sich += verwenden (analog -=, *=, /=).**
- **Zum Erhöhen um 1 kann man ++ schreiben (analog --).**
  x = y + 7 bedeutet, dass x den Wert der Rechnung y + 7 bekommt.
  x = x + 1 bedeutet, dass x um 1 erhöht wird (auch andere Zahlen möglich)
  x += 1 gleichbedeutend: x wird um 1 erhöht (auch andere Zahlen möglich)
  x++ gleichbedeutend: x wird um 1 erhöht (geht nur bei 1)
  x = 2 * x bedeutet, dass x verdoppelt wird.
- Auf der **linken Seite des Gleichheitszeichens steht ausschließlich die Variable, die einen neuen Wert bekommen soll**. Auf der **rechten Seite steht eine Rechnung**, die den neuen Wert ergibt. Dabei darf die Variable links als Wert in der Rechnung rechts vorkommen. Dort wird der alte Variablenwert benutzt.
- Das **Gleichheitszeichen hat eine andere Bedeutung als in der Mathematik**!

1.6.1 Die set-Methoden als Bspl. für verändernde Methode ohne Rückgabewert

Diese einfache Methode verändert den Namen des Kontobesitzers. Eine **Methode ohne Rückgabewert** wird mit **public void** (void entspricht im englischen etwa: nichts) eingeleitet.

```
public void setBesitzerName(String pNeuerName)
{
   besitzerName = pNeuerName;
}
```

1.6.2 Die get-Methoden als Beispiel für sondierende Methoden

Eine ebenso einfache Methode liest den Namen des Kontobesitzers wieder aus:

```
public String getBesitzerName()
{
   return besitzerName;
}
```

Merke: Wann schreibe ich void?

void kommt bei einer Methodendeklaration immer dann vor, wenn es keine Rückgabe gibt.
return Gibt es eine Rückgabe (**return** variable), so muss der Datentyp des Rückgabewertes eingetragen werden.

Übung:	
1)	Schreiben Sie die get- und set-Methode zu der Eigenschaft kontostand.
2)	Schreiben Sie die Methode bucheZinsen (double zinssatz), die auf den vorliegenden Kontostand zinssatz Prozent Zinsen addiert.
3)	Zeichnen Sie das Klassendiagramm zu der aktuellen Klasse Konto.

1.7 Einige Beispiele

Wenn Sie bedenken, wie dick üblicherweise Java-Bücher sind, dann können Sie von den Beispielen zur Zeit noch nicht viel erwarten. Einige einfache Beispiele zeige ich Ihnen hier dennoch, damit Sie einen Überblick bekommen, wie eine Klasse in Java aufgebaut ist.

1.7.1 Der Zähler

Der Zähler ist eine einfache Klasse, bei dem Sie bei Aufruf der Methode erhoeheZaehlerstand() den Zähler um 1 erhöhen. Anwendungen: Verkehrszählungen, Heizungsverbrauchzähler, Bildzähler bei einer Kamera, Rundenzähler bei der Carera-Bahn... . Bei den realen Anwendungen kommt der Impuls von einem Sensor bzw. einen Druckknopf. Bei der Simulation müssen Sie statt dessen eine Methode ausführen.

```
public class Zaehler
{
    // Eigenschaften
    private double zaehlerstand;
    // Konstruktor
    public Zaehler()
    {
       zaehlerstand=0;
    }
    // Methoden
    public void erhoeheZaehlerstand()
    {
       zaehlerstand += 1;
    }
}
```

1.7.2 Das Telefonbuch

Der JavaEditor half bei der Erstellung dieser Klasse. Unter UML | neue Klasse erhalten Sie ein praktisches Auswahlmenu, das Ihnen viel Tipparbeit ersparen kann.
Der **JavaEditor zeigt** übrigens **im Klassendiagramm auch den Konstruktor** an. Das wird in der Literatur nicht einheitlich gemacht, oftmals wird er nicht angegeben.
Die **Telefonnummer** wird, obwohl sie laut Name eine Nummer ist, **als String** eingelesen, um Eingaben wie „/“ zwischen Vorwahl und Nummer zuzulassen.
Die **Besonderheit an dieser Klasse** ist, dass name nur durch den Konstruktor verändert werden kann. D.h. ich kann zwar mit steTelefonnummer nachträglich Telefonnummern ändern, aber der Name bleibt erhalten. Allernfalls durch das Löschen des Objektes kann ich Personen aus dem Telefonbuch entfernen.
Achtung: Ohne dass wir in Dateien speichern können, werden bei einem **Neustart von BlueJ alle Objekte wieder entfernt**. Geben Sie also nicht die Telefonnummern ihrer gesamten Freunde ein, das Telefonbuch ist noch nicht fertig!

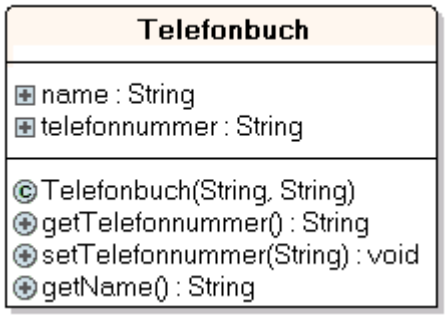


Abbildung 6: Klassendiagramm Telefonbuch

```
public class Telefonbuch {

  // Eigenschaften
  private String name;
  private String telefonnummer;

  // Konstruktor
  public Telefonbuch(String pName, String pTelefonnummer) {
  // Hier fehlen 2 Zeilen. Bitte nachtragen! (vgl. Übung)
  }
```

```
  // Methoden
  public String getTelefonnummer() {
    return telefonnummer;
  }

  public void setTelefonnummer(String pTelefonnummer) {
    telefonnummer = pTelefonnummer;
  }

  public String getName() {
    return name;
  }

  // Ende Ereignisprozeduren
}
```

1.7.3 Das Auto

Die folgende Klasse ist schon sinnvoller. Sie ist in ähnlicher Weise in vielen Bordcomputern von modernen Autos eingebaut: Die elementare Frage beim Autofahren ist: wie weit komme ich noch, ohne zu tanken.
Die Klasse simuliert ein Auto, erfasst aber nur Daten um den Tank und den Verbrauch.
Besonderheit: **tanke(..)** und **fahre(...)** sind Methoden, die den aktuellen Füllstand beeinflussen. Es sind auch in der Realität die einzigen Arten, wie man den Füllstand eines Autos ändert (von klaueBenzin(), tropfeHeraus() und verdampfe() abgesehen).
Spannend ist die sondierende Methode **getAktuelleMaximaleReichweite**, die den Wert in km zurückgibt, der bei aktuellem Füllstand noch gefahren werden kann.
Die Probleme der Klasse liegen noch darin, dass soviel tanken können wie wir wollen (auch über das Tankvolumen hinaus) und dass wir den Tankinhalt auch in den negativen Bereich fahren können (Dispokredit bei der Tankstelle).
Das können wir erst lösen, wenn wir die bedingte Ausführung kennen gelernt haben.
Der folgende Quelltext ist aus dem JavaEditor übernommen.

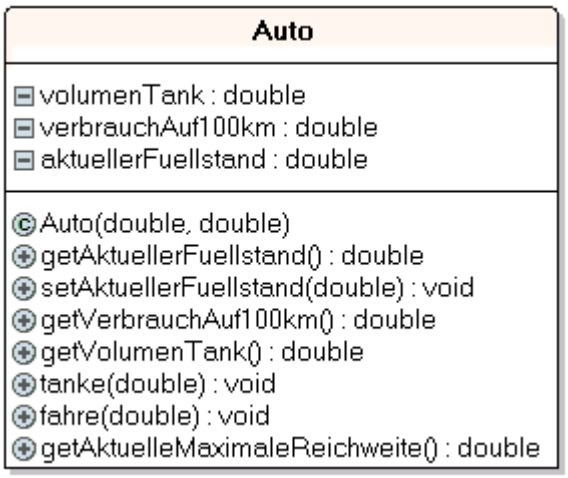


Abbildung 7: Klassendiagramm des Autos, bei dem nur die Tankdaten interessieren

```
public class Auto {

  // Anfang Variablen
  private double volumenTank;
  private double verbrauchAuf100km;
  private double aktuellerFuellstand;
  // Ende Variablen

  // Anfang Ereignisprozeduren

  // Konstruktor
  public Auto(double pVolumenTank, double pVerbrauchAuf100km) {
    volumenTank=pVolumenTank;
```

```
    verbrauchAuf100km=verbrauchAuf100km;
    aktuellerFuellstand=0; // Bei Auslieferung ist der Tank leer
  }

  public double getAktuellerFuellstand() {
    return aktuellerFuellstand;
  }

  public void setAktuellerFuellstand(double pAktuellerFuellstand) {
    aktuellerFuellstand = pAktuellerFuellstand;
  }

  public double getVerbrauchAuf100km() {
    return verbrauchAuf100km;
  }

  public double getVolumenTank() {
    return volumenTank;
  }

  public void tanke(double pMenge) {
    aktuellerFuellstand+=pMenge; // fuege Menge hinzu
  }

  public void fahre(double pKilometerZahl) {
    // Die Formel sagt folgendes aus:
    // vom Fuellstand wird der Verbrauch bei der Kilometerzahl abgezogen
    aktuellerFuellstand -= pKilometerZahl / 100 * verbrauchAuf100km;
  }

  public double getAktuelleMaximaleReichweite() {
    return aktuellerFuellstand / verbrauchAuf100km * 100;
  }

  // Ende Ereignisprozeduren
}
```

Übung:	
1)	Analysieren und testen Sie alle 3 Programme. Warum bekommt nicht jede Eigenschaft eine get- und eine set-Methode?
2)	Zu Zähler: a) Ergänzen Sie die Methode: getZaehlerstand() b) Ergänzen Sie die Methode: aufNullSetzen() und erhöheUm(double pZahl)
3)	Zu Telefonbuch: Ergänzen Sie die fehlenden Zeilen zum Konstruktor. Ergänzen Sie die Eigenschaften geburtsdatum (ggf. noch schuhgröße). Welchen Datentypen verwenden Sie? Benötigt die Eigenschaft eine set-Methode?
4)	Zum Auto: Ergänzen Sie die Methode: getMaximaleReichweiteBeiVollemTank()
5)	Entwickeln Sie die Klasse Handy, die die Eigenschaften telefonnummer, guthaben und gespraechskostenProMinute. Entwickeln Sie geeignete Methoden.
6)	Übung für Fortgeschrittene: Realisieren Sie die Klasse Bruch mit den Eigenschaften

Übung:	
	zaehler und nenner und realisieren Sie die Methoden addierenEinerGanzenZahl, kuerzen und erweitern. Wenn Sie schon Vorwissen haben, können Sie hier für eine erweiterte Variante schauen: http://www.u-helmich.de/inf/BlueJ/exkurse/bruchrechnung.html

1.8 Interaktion zwischen Objekten

Zum Schluss des Kapitels behandeln wir, wie zwei Objekte miteinander kommunizieren können. Ein schönes Beispiel dafür die die Methode ueberweise().

```
public void ueberweise(double pBetrag, Konto pZielkonto)
{
   abheben(pBetrag);
   zielkonto.einzahlen(pBetrag);
}
```

Erläuterung: Es gibt diesmal zwei Parameter, die wir, wie schon beim Konstruktor, mit Komma trennen. Die Besonderheit ist, dass der Datentyp von Zielkonto wieder ein Konto ist. Man darf also Klassen als Datentypen angeben – ja, man darf sogar in Methoden den Datentypen der Klasse verwenden, zu dem die Methode gehört. Das ist sinnvoll, denn üblicherweise möchte man von einem Konto auf ein anderes Konto überweisen. Also muss das Konto überweisen können (wenn man keine Klasse Bank erschafft, die die Konten verwaltet. Aber das ist ein Thema für später).

```
   abheben(betrag);
   zielkonto.einzahlen(pBetrag);
```

Wichtig!
Eigenschaften, die private sind, sind nur für Objekte der gleichen Klasse sichtbar.
D.h. theoretisch können wir von dem einen Konto auf die Eigenschaften des anderen Kontos zugreifen. Damit es nicht zu Verwechselungen kommt, sollten Sie sich das aber erst gar nicht angewöhnen.
Konvention:
Auf die **Daten von fremden Objekten oder Klassen greifen wir immer über Methoden** und nie über deren Eigenschaften zu. Dies gewährleistet, dass nichts Ungewolltes verändert wird (vgl. Waschmaschinenbeispiel).
Nur innerhalb des Objektes selbst verändert man die eigenen Eigenschaften direkt.

Übung:	
1)	Schreiben Sie die Methode holeEinzug (double betrag, Konto zielkonto), bei dem das Geld von einem fremden Konto überwieden wird.
2)	Schreiben Sie die Methode vererben (Konto erbe), bei dem das gesamte Geld eines Kontos auf ein anderes verbucht wird.
3)	Schreiben Sie die Methode binIchReich(Konto vergleichskonto), das die Differenz aus dem eigenen Kontostand mit einem anderen Konto vergleicht.
4)	Schreiben Sie die Methode stifteVerwirrung(Konto konto1, Konto konto2), bei ein unbeteiligtes Konto dafür sorgt, dass das gesamte Geld von konto1 auf konto2 überwiesen wird.

Übung:	
5)	Schreiben Sie die Methode stifteNochMehrVerwirrung(Konto konto1, Konto konto2), bei ein unbeteiligtes Konto dafür sorgt, dass das gesamte Geld von konto1 halbiert, das von konto2 aber verdoppelt wird.
6)	Schreiben Sie die Methode bekommeZinsen(double zinsatz), das entsprechend des Zinssatzes den Kontostand erhöht.
7)	Lassen Sie von einem Konto den Namen des Besitzers eines anderen Kontos ändern. Welche Probleme ergeben sich, wenn Sie die Methode setBesitzerName weglassen?

1.9 Anhang A – Exkurs: Überweisen in 3 Varianten

```
public void ueberweise(double betrag, Konto zielkonto)
  {
     double zwischenspeicher;
     kontostand -= betrag;
     // Um den Betrag zu buchen, schlage ich 3 Varianten vor.
     // Bitte entfernen Sie die Kommentierung, wenn Sie eine andere Variante
     // testen wollen.

     // *** Variante 1: ***
     // zielkonto.einzahlen(betrag); // diese Variante ist sehr effektiv.
     // Sie nutzt die Methode einzahlen des Zielkontos,
     // lässt aber keinen Zugriff per if auf Kontostände des Zielkontos zu.
     // daher eine zweite Variante, die auch funktioniert:
     // *** Variante 2:  ***
     zwischenspeicher = zielkonto.getKontostand();
     zwischenspeicher += betrag;
     zielkonto.setKontostand(zwischenspeicher);
     // oder auch in kompakter Form ohne Zwischenspeicher
     // *** Variante 3: ***
     // zielkonto.setKontostand(zielkonto.getKontostand()+betrag);
  }
```

1.10 Anhang B – Die Online-Bank Quelltext komplett

```
public class Konto {
  // (Instanzvariablen der) Eigenschaften
  private String besitzerName;
  private double kontostand;

  //Konstruktor
  public Konto (String pBesitzerName){
    besitzerName = pBesitzerName;
    kontostand = 0.0;
  }

  // Methoden
  public void setBesitzerName(String pNeuerName)
  {
     besitzerName = pNeuerName;
```

```
   }

   public String getBesitzerName()
   {
      return besitzerName;
   }

   public void setKontostand(double pBetrag)
   {
      kontostand = pBetrag;
   }

   public double getKontostand()
   {
      return kontostand;
   }

   public void einzahlen(double pBetrag)
   {
      kontostand += pBetrag;
   }

   public void ueberweise(double pBetrag, Konto pZielkonto)
   {
      kontostand -= pBetrag;
      zielkonto.einzahlen(pBetrag);
   }

} // Diese Klammer muss am Ende stehen
```

1.11 Weblinks

- BlueJ Lehrgang: http://www.u-helmich.de/inf/BlueJ/kurs11/seiten/seite04.html
- BlueJ Lehrgang kompakt: http://www.informatik.ursulaschule.de/index.php?cid=368
- Java Kurs: http://www.uni-muenster.de/ZIV/Mitarbeiter/BennoSueselbeck/java/ss2004/obj/obj.html
- Java Lehrgang (Objekt): **http://www.boku.ac.at/javaeinf/jein1.html#object**
- http://de.wikipedia.org/wiki/Objektorientierte_Programmierung und http://de.wikipedia.org/wiki/Objektdiagramm, Klassendiagramm: http://de.wikipedia.org/wiki/Klasse_%28UML%29
- Das Klassendiagramm wurde mit dem Java-Editor erstellt: siehe http://www.zum.de/wiki/index.php/Java-Editor