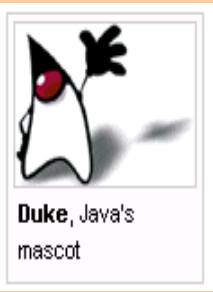


Der **Star Seven** (im Original: *7) war ein im Rahmen des „Green Project“ bei Sun Microsystems entwickelter, portabler Minicomputer, der zur Steuerung von Haushaltsgeräten dienen sollte. Nach seiner internen Vorstellung im Herbst 1992 wurde die Firma First Person, Inc. gegründet, die die Vermarktung übernehmen sollte. Das Gerät gelangte allerdings nie auf den Markt. Bereits 1994 galt die Hardware als obsolet, und man machte sich daran die vielversprechendsten Komponenten (Java) zu eigenständigen Produkten weiterzuentwickeln. Bestandteil des Systems waren das Betriebssystem Green-OS, der Interpreter Oak (später Java) und einige Hardwarekomponenten. Der User interagierte mit dem System über eine grafische Oberfläche, die mit Hilfe von grafischen Assistenten durch die Dialoge führte. Einer dieser Assistenten war der Duke, der sich heute als Maskottchen von Java wiederfindet.
http://de.wikipedia.org/wiki/Star_Seven - http://en.wikipedia.org/wiki/Java_programming_language

Einführung in die Objektorientierte Programmierung mit Java

Kapitel 1 – Erste Schritte

Inhaltsverzeichnis

"Als es noch keine Computer gab,
war das Programmieren noch relativ einfach."
http://de.wikiquote.org/wiki/Edsger_W._Dijkstra
Edsger Wybe Dijkstra (1930-2002) war ein einflussreicher niederländischer Informatiker.

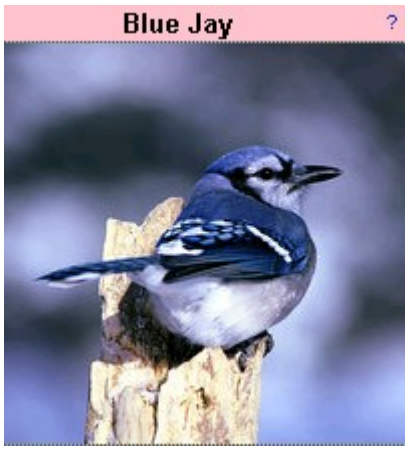
1 Objektorientierte Programmierung

Lernziele

Sie sollen kennen lernen,
- was ein Objekt und eine Klasse sind
- was eine Eigenschaft und eine Methode eines Objektes sind
- was ein Klassen- und Objektdiagramm ist

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

Zunächst klären wir, was ein Objekt ist:
Vom Alltagsverständnis her ist ein **Objekt** ein beliebiger Gegenstand, den man beschreiben kann unter eventuell selbst etwas machen kann. Das ist in Informatik ganz ähnlich.
Nehmen wir beispielsweise das Objekt Vogel:



Ein Vogel hat bestimmte **Eigenschaften (oder Attribute)**. Er hat zum Beispiel eine Farbe, ist männlich oder weiblich und man kann seine Flügelspannweite messen.
Ein konkreter Vogel, ein Objekt Vogel, hat einen bestimmten **Zustand**: ein blauer männlicher Vogel, der eine Spannweite von 20 cm hat (zum Beispiel ein Blue Jay).
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(...)).
Die Objekte in Informatik können Auskunft über sich selbst geben (**beobachtende** oder auch **sondierende Methoden**). So können sie nach ihren Eigenschaften fragen. Dazu haben sie in der Regel Methoden, die die Eigenschaften abfragen: *gibFarbe*() oder *getFarbe*().
Zudem haben sie Methoden, die ihre Eigenschaften ändern können (**verändernde Methode**). Ein Beispiel wäre *setFarbe("blau").*
Objekte vom Typ Vogel fasst man unter einer **Klasse** zusammen. Eine Klasse ist vergleichbar mit einem Bauplan eines Objektes. Zum Bauen des Vogels muss man seine Eigenschaften und seine Methoden kennen.

- **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** 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

Die Stempel Analogie
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

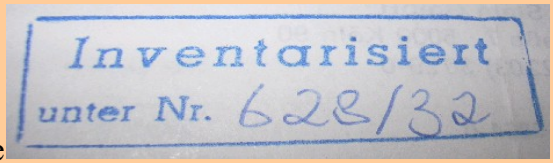


Abbildung 2: Diese Klasse Stempel hätte die Eigenschaft inventarnummer. Der Zustand des Objektes stempel1 ist so, dass die Inventarnummer 628/32 ist.

Die Autoquartett Analogie
Sie kennen alle die Quartettspiele, die Kinder (zumindest kleine Jungs) gerne spielen: Sie haben ein Bild eines Autos und darunter verschiedene Daten. Auch dies ist ein passendes Beispiel für Klassen und Objekte: Jede Karte ist ein Objekt, auf jeder Karte ist ein unterschiedliches Auto mit unterschiedlichen Daten (Porsche 200PS, Ente 30PS), d.h. die Zustände der Objekte unterscheiden sich (sonst wäre das Spiel langweilig). Als Klasse kann man das Grundmuster der Karten ansehen: Die Karten müssen die gleichen Eigenschaften benutzen, sonst lassen sich die Daten nicht vergleichen. Die Klasse Autokarte kann z.B. die Eigenschaften ps, gewicht und baujahr umfassen.

Übung:
1) Was sind die Eigenschaften der Klasse Kartenspiel (Skatspiel)?
2) Nennen Sie Eigenschaften und Methoden des Objektes Kaffeemaschine.
3) Recherchieren Sie im Internet nach griffigen Definitionen für Objekte, Klassen. Methoden und Eigenschaften.

1.1.1 Klassendiagramm und Objektdiagramm

Im **Klassendiagramm** hält man die Elemente einer Klasse anschaulich fest: Klassenname, Eigenschaften und Methoden. Sie sehen hier das Klassendiagramm der Klasse mit dem Namen Vogel. Zur Vereinfachung wurden nur die Eigenschaft *farbe* und die Methoden getFarbe() und setFarbe() aufgeführt. + bedeutet, dass die Methode public, also öffentlich zugänglich ist. -, dass die Eigenschaft private ist, also nur innerhalb des Objektes selbst ausgelesen werden kann. Wir setzen Eigenschaften immer auf private und Methoden immer auf public. Die Regel genügt zunächst, damit Sie ein Grundverständnis haben.
Im **Objektdiagramm** hält man ein spezielles Objekt fest, was aus einer Klasse gebildet wurde. Es beinhaltet nur den Objektnamen und die Attribute.

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

1.1.2 Klassendiagramm Vogel

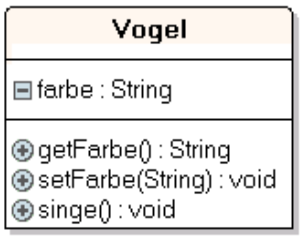


Abbildung 3: Klassendiagramm Vogel

Drei Beispiele für Objektdiagramme

donald:Vogel
farbe: weiß

bibo:Vogel
farbe: gelb

blue_Jay:Vogel
farbe: blau

1.1.3 Auswahl der Eigenschaften und Methoden

Welche Eigenschaften und Methoden Sie für eine Klasse benutzen, hängt von Ihrem Anwendungsfall ab. Nehmen wir als Beispiel einen Monitor. Wenn ich eine Treibersoftware für einen Monitor entwerfe, so ist mir die Farbe des Gehäuses egal, relevant sind technische Daten wir Auflösung und Betriebsspannung. Anders ist es, wenn ich eine Software für die Lagerhaltung eines Elektrofachhandels schreibe: die Kunden möchten dann wissen, wie ihr Monitor aussieht. Relevant sind Daten wie Farbe, Hersteller und Preis des Monitors. Entsprechend variiert meine Klasse Monitor je nach Anwendungsfall.

Übung:
1) Legen Sie ein Klassendiagramm und drei Objektdiagramme zu einer Kaffeemaschine an.
2) Legen Sie ein Klassendiagramm zu einem beliebigen Ding an.

1.2 Turmbau zu Babel und seine IT-Analogie


Meister der Weltenchronik

*„Die Erzählung vom **Turmbau zu Babel** (Genesis 11,1-9) beschließt die biblisch-mythischen Erzählungen des Buchs Genesis.(...) In Folge des Turmbaus verwirrte Gott die Sprache der Menschen, die den Turm bauten und zerstreute sie über die ganze Erde.“*
http://de.wikipedia.org/wiki/Turmbau_zu_Babel (auch das Bild)
Wahrscheinlich ist es in der Informatik nicht eine Strafe Gottes, aber eine einheitliche Schreibweise oder Ausdrucksweise besteht nicht. So existieren dutzende Programmiersprachen nebeneinander, die Bezeichnung von Fachbegriffen selten eindeutig. Programme der selben Sprache laufen noch lange nicht auf einem PC und einem Mac. Auf dem PC gibt es Linux und Windows, die beide unterschiedliche Programmvarianten benötigen. Selbst Vereinheitlichungen wie UML-Diagramme werden nicht einheitlich genutzt. Die Konsequenz für diese Sammlung ist, dass Ihnen verschiedene Ausdrucksweisen für die selben Fachbegriffe vermittelt werden (Eigenschaft und Attribut, Zustand und Status).
Verschiedene Programmiersprachen vorzustellen, wäre für einen Anfänger fatal. Es lohnt sich aber ein Blick auf diese Seite: http://de.wikipedia.org/wiki/Hallo_Welt
Java ist eine Sprache, die zumindest **plattformübergreifend** ist. Java Programme laifen unter

Linux, Windows und MacOS problemlos. Sogar auf dem Handy lassen sich JavaProgramme ausführen, wenn diese mit einem speziellen Compiler übersetzt wurden.
UML ist zwar wieder eine Sprache, aber sie wird dazu genutzt, Programme und Programmstrukturen zu planen. In UML geplante Programme lassen sich einfach von einer Sprache zu einer anderen übertragen, da UML die Grundstruktur vorgibt.

1.3 Exkurs: Was ist UML

*„Klassen und Objektdiagramme sind zwei Diagrammarten von UML. Die **Unified Modeling Language (UML)** ist eine von der Object Management Group (OMG) entwickelte und standardisierte Sprache für die Modellierung von Software und anderen Systemen. Im Sinne einer Sprache definiert die UML dabei Bezeichner für die meisten Begriffe, die für die Modellierung wichtig sind, und legt mögliche Beziehungen zwischen diesen Begriffen fest. Die UML definiert weiter **grafische Notationen für diese Begriffe** und für Modelle von statischen Strukturen und von dynamischen Abläufen, die man mit diesen Begriffen formulieren kann.*

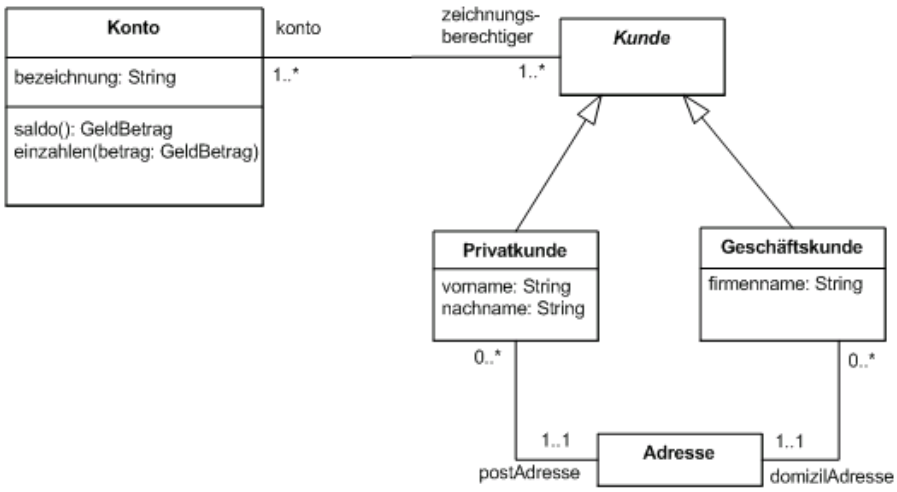


Abbildung 4: Dieses Klassendiagramm besteht aus mehreren Klassen, die miteinander in Beziehung stehen

*Die **UML ist heute eine der dominierenden Sprachen für die Modellierung von Softwaresystemen**. Die meisten Leute kommen zum ersten Mal mit der UML in Kontakt, wenn sie in der einen oder anderen Rolle an einem Softwareprojekt beteiligt sind. Der erste Kontakt besteht häufig darin, dass gewisse Diagramme der UML zu erstellen, zu verstehen oder zu beurteilen sind.*“
Siehe http://de.wikipedia.org/wiki/Unified_Modeling_Language
Ohne es genau zu erläutern sehen Sie hier, wie ein Klassendiagramm aussieht, bei dem die Beziehungen zwischen verschiedenen Klassen eingezeichnet sind.

2 Java mit BlueJ - Beispiel Onlinebank

Auf dieser Seite sollen Sie anhand der Simulation eines Online-Kontos selbst Klassen definieren und die grundlegenden Elemente einer (objektorientierten) Programmiersprache kennen lernen

- Eine Klasse umgesetzt in Java-Quelltext
- Datentypen für Zahlen: double, für Zeichenketten: String
- Instanz eines Objektes, Der Objektinspektor von BlueJ, Kommentare im Quelltext
- Eigenschaften und Methoden, der Konstruktor als spezielle Methode

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].

2.1 Die Klasse Konto als Beispiel für einen ersten Quelltext

Wie sieht eine Klasse Konto als Quelltext aus (z.B. Girokonto an einer Online-Bank)? Um diese Frage zu klären, überlegen wir uns, welche Anforderungen ein Auftraggeber, der ein Kontoverwaltungsprogramm wünscht, stellen würde.
Eine Online-Bank benötigt zumindest den Namen des Besitzers und den aktuell vorhandenen Geldbetrag, um ein Konto anzulegen.
besitzerName und *kontostand* sind 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**. Für unseren Zweck sind zwei geeignet: **String** und **double**.
Zeichenketten, also Aneinanderreihungen von Zeichen, werden in Java durch dem Datentypen String deklariert. Der **Datentyp String** eignet sich als Datentyp für *besitzerName*, da dieser aus beliebigen Zeichen der Tastatur[2] besteht und keine Zahl ist.
Als Typ für den Kontostand *kontostand* wird ein numerischer Datentypen gewählt. **double** kann Zahlen im Bereich von +/-1,7E+308 (also 17 mit 307 Nullen) abspeichern, was für unseren Zweck genügen sollte[3].

Zwei Datentypen: String und double
- **String** kann Zeichenketten speichern.
- **double** speichert Zahlen im Bereich von +/-1,7E+308 (also 17 mit 307 Nullen).

Hintergrundinformationen: http://de.wikipedia.org/wiki/Doppelte_Genauigkeit
http://de.wikipedia.org/wiki/Zeichenkette

Übung:
1) Nennen Sie Anwendungen, bei denen der Bereich von double nicht genügt.

2.1.1 Der erste Quelltext der Klasse Konto

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***.

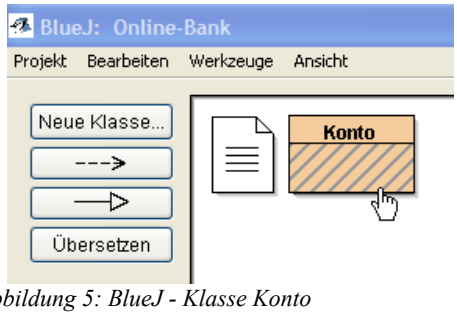


Abbildung 5: BlueJ - Klasse Konto

public und **private** regeln die Zugriffsrechte

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)
2 Um genau zu sein: Es können sogar Zeichen dargestellt werden, die nicht mit der Tastatur zu erreichen sind (z.B. ausländische Sonderzeichen).
3 Zu beachten ist, dass eine **Gleit- oder Fließkommazahl** wie double nicht in seinem gesamten Datenbereich genau ist, was für ein professionelles Programm gegen den Datentypen sprechen würde. Eine Zahl mit 308 Stellen würde gerundet werden. Auf der anderen Seite benötigen wir für Finanzmathematik 2 Nachkommastellen und ggf. 10 Stellen vor den Komma. In diesem Bereich geschehen - von Rundungsproblemem abgesehen - keine Fehler. Zur genauen Recherche siehe http://de.wikipedia.org/wiki/Flie%C3%9Fkommazahl

(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).

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

Nun kommt BlueJ[4] ins Spiel. Mit "Projekt|neu" wird ein neues Projekt angelegt, das wir "Online-Bank" nennen wollen. Mit "Neue Klasse" legen wir eine Klasse mit dem Namen "Konto" an. Wenn alles richtig geklappt hat, sollte es etwa so aussehen wie in Abbildung 5.

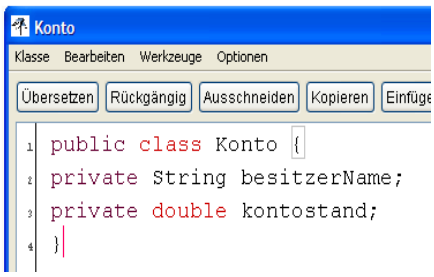


Abbildung 6: Editor zur Quelltexteingabe

Mit einem Doppelklick auf das beige Rechteck kommt man zum Editor, in dem sich der Quelltext eintippen lässt. Anschließend kann mit dem Button "Übersetzen" abspeichert und "compiliert" werden (Abb. 6).

Unten am Editorfenster sollte nun der folgende Text zu sehen sein "Klasse übersetzt - keine Syntaxfehler" (Abb. 7). Wenn das nicht der Fall ist, beginnt die Suche nach Tippfehlern.

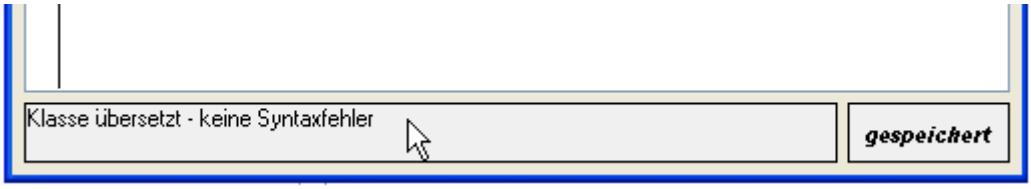


Abbildung 7: Gespeichert, Compiliert und ohne Fehler

Nun sollten in dem Hauptfenster von BlueJ die Streifen von der Klassendarstellung verschwunden sein. Mit Hilfe der rechten Maustaste kann ich ein "*Objekt Konto*"[5] anlegen, indem ich new Konto() wähle.

Abbildung 8: Anlegen eines neuen Objektes aus einer Klasse

Abbildung 9: Der Objektinspektor

ın, allerdings stirbt diese präzise Sprechweise

Ich wähle den vorgegebenen Namen *konto1* und habe mein Objekt *konto1 vom Typen Konto*, welches BlueJ unten links als rotes Rechteck visualisiert (konto1:Konto vgl. Abb.). Jedes Objekt bekommt einen Namen, da es unterscheidbar sein muss. Legt man ein weiteres Objekt vom Typ Konto an, so schlägt BlueJ als Namen *konto2* vor, wir können es aber auch anders nennen. Unzulässig ist, einen Namen doppelt zu verteilen.

Ein Rechtsklick auf das rote Rechteck und dann auf Inspizieren öffnet bei BlueJ den **Objektinspektor** (Abb. 9). 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.

2.2 Kommentare – Quelltexte verständlich machen

- Zur besseren **Verständlichkeit des Quelltextes können Kommentare gesetzt werden**. Der Computer ignoriert die Zeile bei der Ausführung.
- Kommt in einer Zeile // vor, so ist dies ein Kommentar und das dahinterstehende wird nicht ausgeführt. Bei BlueJ lässt sich mit F8 ein markierter Bereich in Kommentare setzen. F7 hebt die Kommentierung wieder auf. Dies lässt sich auch im Menu „Bearbeiten“ aufrufen.
- Mehrere Zeilen lassen sich mit /* und */ auskommentieren
- Bestimmte Kommentare werden von Java genutzt, um eine Dokumentation einer Klasse aufzubauen. diese Kommentare werden mit /** und */ gesetzt.

```
// Kommentar

/*  public double getAktuelleMaximaleReichweite() {

    ...

*/
```

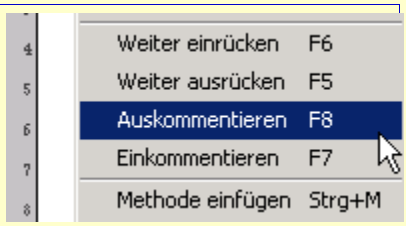


Abbildung 10: BlueJ: Kommentierung im Bearbeiten-Menu

2.3 Zusammenfassung

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

Übung:	
1)	Erweitern Sie die Klasse Konto um die Attribute *kreditlimit* und *telefonnummer*. Welche Datentypen würden Sie verwenden?

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

2.4 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**. Dabei besteht in der Regel die Möglichkeit, unterschiedliche Initialisierungsvarianten mittels der Aufrufparameter des Konstruktors auszuwählen.
Der Destruktor wird dagegen bei der Zerstörung der Variablen aufgerufen, und erledigt ggf. Aufräumarbeiten. Dieser Vorgang heißt dann Deinitialisierung.
Nach: http://de.wikipedia.org/wiki/Konstruktor

Beispiel Konto

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

  //Konstruktor
  public Konto (String besitzerName){
    this.besitzerName = besitzerName;
    this.kontostand = 0.0;
  }
}
```

Die Klasse Konto hat nun einen Konstruktor bekommen. In Java wird dieser in den Quelltext der Klasse eingefügt. Er wird mit **public Klassenname** eingeleitet. Achtung: im Gegensatz zur Zeile 1 wird hier **nicht das Wort class** verwendet.

Welche Funktion hat der Konstruktor der Klasse Konto?
Fangen wir hinten an: `this.kontostand = 0.0;` setzt die Eigenschaft kontostand auf 0. Falls ich möchte, dass ein neu angelegtes Konte zumindest einen Euro beinhaltet, schreibe ich einfach: `this.kontostand = 1.0;`
Das 1.0 schreibe ich hier bewusst, um zu verdeutlichen, dass kontostand vom Datentyp double ist und somit Nachkommastellen zulässt. Zudem sollten Sie sich möglichst früh an die amerikanische Zahlennotation gewöhnen:

Die Zahl **7,34 wird in Java 7.34** geschrieben
(Es gilt die **amerikanischeZahlennotation: Punkt statt Komma**)

Schwieriger wird es, die Befehle um besitzerName zu verstehen.

```
  public Konto (String besitzerName){
```

besitzerName wird als Parameter übergeben. Beim Aufruf des Konstruktors hat lässt sich der Wert

angeben (BlueJ macht dies besondres komfortabel: siehe unten)

```
this.besitzerName = besitzerName;
```

Diese Zeile bedeutet folgendes: die Eigenschaft der derzeitigen Klasse (also Konto) besitzerName bekommt den Wert der Varible besitzerName zugewiesen.

Das Schlüsselwort this

this.*eigenschaftsname* ergibt den Wert der Eigenschaft des aktuellen Objektes. Innerhalb einer Klasse kann man auch mit *eigenschaftsname* die Eigenschaft abfragen, allerdings nur, wenn nicht der Eigenschaftsname gleichzeitig als Variablenname in einer Methode verwendet wird. In diesem Fall muss man das **this** benutzen, damit für Java eindeutig geklärt ist, dass die Eigenschaft gemeint ist.

Typischer Anwendungsfall: Konstruktor

Hier wird absichtlich ein mit der Eigenschaft identischer Variablenname gewählt, um zu verdeutlichen, dass der Übergabeparameter des Konstruktors letztlich zum Wert der Eigenschaft wird.

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

  //Konstruktor
  public Klassenname (String xyz){
    this.xyz = xyz;
    this.kontostand  = 0.0;
  }
}
```

Alternativ lassen sich auch verschiedene Variablennamen verwenden. Die folgende Schreibweise ist gleichwertig zu der oben abgebildeten. In diesem Fall kann man auf das **this** verzichten (es funktioniert aber auch, wenn es dort stehen bleibt).

```
public class Klassenname {
  // (Instanzvariablen der) Eigenschaften
  private String xyz;

  //Konstruktor
  public Klassenname (String parameterxyz){
    xyz = parameterxyz;
    kontostand = 0.0;
  }
}
```

Langer Rede kurzer Sinn: Beim Konstruktor ist eine Parameterübergabe möglich.
Bei BlueJ sieht das dann praktisch angewendet so aus:

Abbildung 12: Mit der rechten Maustaste rufe ich das Kontextmenu zu Konto auf, dann erzeuge ich ein Objekt Kont, indem ich auf new Konto(...) klicke.

Abbildung 11: Dazu werde ich gefragt, welchen Wert der Parameter besitzerNamen erhalten soll (Anführungsstriche wegen String!)

Abbildung 13: Der Objektinspektor (im Kontextmenu des roten Objektdiagramms von konto1) zeigt mir, dass besitzerName nun Dagobert ist. Der Konststand ist wunschgemäß 0.

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).

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

  //Konstruktor
  public Konto (String besitzerName, String strassenName){
    this.besitzerName = besitzerName;
    this.strassenName = strassenName;
    this.kontostand = 0.0; // identisch mit kontostand=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 und hausnummer*. Überlegen Sie dabei, welche Datentypen Sie verwenden (String oder double)!

2.5 Methoden – Was kann meine Klasse?

Zuletzt fehlen für ein vollständiges Objekt nur noch Methoden. In Wikipedia steht:

„Die **einer Klasse (Informatik) von Objekten zugeordneten Algorithmen** bezeichnet man auch als **Methoden**."
aus: http://www.zum.de/wiki/index.php/Methode_%28Informatik%29

Das hilft uns nicht weiter. Zunächst müssen wir klären, was ein Algorithmus ist:

Unter einem **Algorithmus** versteht man allgemein eine **genau definierte Handlungsvorschrift zur Lösung eines Problems** oder einer bestimmten Art von Problemen.
Im täglichen Leben lassen sich leicht Beispiele für Algorithmen finden: Zum Beispiel ist ein **Kochrezept** ein Algorithmus – zumindest dann, wenn alle Angaben genau genug sind und es für alle Teilaufgaben, wie Braten, Rühren, etc., ebenfalls Algorithmen gibt. Auch **Reparatur- und Bedienungsanleitungen** oder Hilfen zum Ausfüllen von Formularen sind in der Regel Algorithmen. Ein weiteres, etwas präziseres Beispiel sind Waschmaschinenprogramme.
http://de.wikipedia.org/wiki/Algorithmus

Zwei Standardmethoden sind von herausragender Bedeutung:

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

Eine einfache Methode verändert den Namen des Kontobesitzers:

```
public void setBesitzerName(String neuerName)
{
   besitzerName = neuerName; // oder  this.besitzerName = neuerName;
}
```

`setBesitzerName` ist der Name der Methode. set deutet dabei an, dass die Methode eine Eigenschaft ändert.
Es wird ein neuer Name als Parameter übergeben `setBesitzerName(String neuerName)`
Dieser wird der Eigenschaft besitzerName zugewiesen: `besitzerName = neuerName;`

Abbildung 14: Im Kontextmenu des Objektes konto1 taucht nach Neucompilierung eine neue Zeile auf: Unsere Methode setBesitzerName(...)

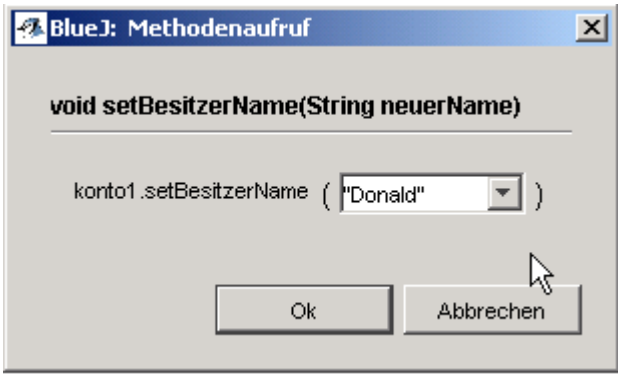


Abbildung 15: Führe ich sie aus, kann ich den Besitzernamen ändern. Eine offene Frage ist allerdings: Was sagt Dagobert dazu?

2.5.2 Die get-Methoden als Beispiel für sondierende Methoden

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

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

// oder gleichbedeutend
```

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

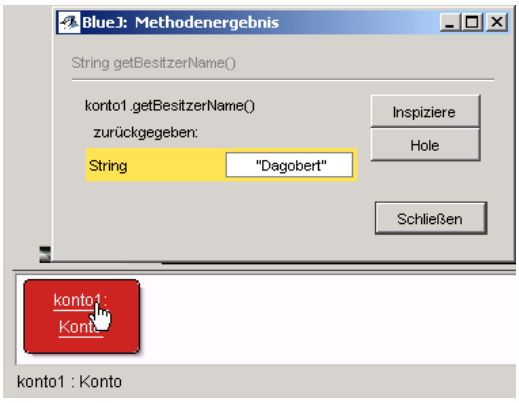


Abbildung 16: Aufruf der Methode getBesitzerName() - BlueJ zeigt das zurückgegebene Methodenergebnis in einem Fenster an.

Eine **Methode ohne Rückgabewert** wird mit **public void** (void entspricht im englischen etwa: nichts) eingeleitet.
Eine **typische Methode dieser Klasse ist sie verändernde Methode**, die eine Eigenschaft verändert. Methoden dieser Klasse werden üblicherweise mit **set** als Ergänzung zum Methodennamen benannt.

```
public void setEigenschaft(Datentyp variable)
{
   this.eigenschaft = variable;
}
```

Eine **Methode mit Rückgabewert** wird mit **public *Datentyp*** eingeleitet.
Eine **typische Methode dieser Klasse ist die sondierende Methode**, die eine Eigenschaft ausliest. Methoden dieser Klasse werden üblicherweise mit **get** als Ergänzung zum Methodennamen benannt. Der Befehl **return** vollzieht dabei die Rückgabe des Wertes.

```
public Datentyp getEigenschaft()
{
   return this.eigenschaft;
}
```

Diese beiden Methodenarten sind so elementar, dass der JavaEditor sie sofort automatisch hinzufügt, wenn dies unter Optionen so eingestellt wird.

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.

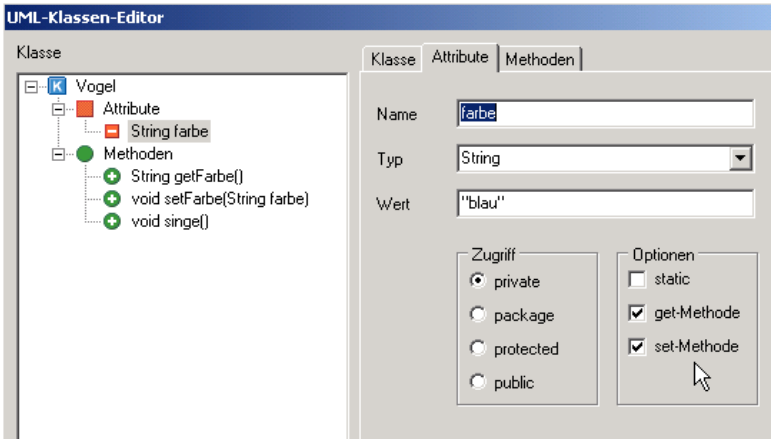


Abbildung 17: JavaEditor legt selbst get- und set-Methoden an.

2.5.3 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 betrag)
{
   this.kontostand += betrag;
   // oder this.kontostand = this.kontostand + betrag;
}
```

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

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 wird völlig anders verwendet als in der Mathematik!

Übung:	
1)	Schreiben Sie die get- und set-Methode zu der Eigenschaft kontostand.
2)	Schreiben Sie die Methode abheben(...), die einen Betrag vom Konto abhebt.
3)	Schreiben Sie die Methode abraeumen( ), die einen Kontostand wieder auf 0 setzt.
4)	Schreiben Sie die Methode bucheZinsen (double zinssatz), die auf den vorliegenden Kontostand zinssatz Prozent Zinsen addiert.
5)	Zeichnen Sie das Klassendiagramm zu Konto sowie 3 Objektdiagramme.

2.6 *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.

2.6.1 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
{
    // Instanzvariablen der Eigenschaften
    private double zaehlerstand;
    // Konstruktor
    public Zaehler()
    {
        // Der Anfangszaehlerstand soll immer 0 sein.
        zaehlerstand=0;
    }
    / Methoden
    public void erhoeheZaehlerstand()
    {
        zaehlerstand += 1;
    }
}
```

2.6.2 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.

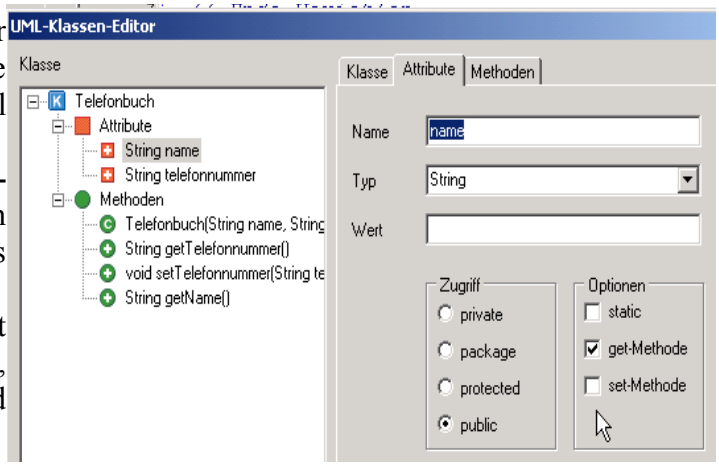


Abbildung 18: JavaEditor erspart Tipparbeit

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!

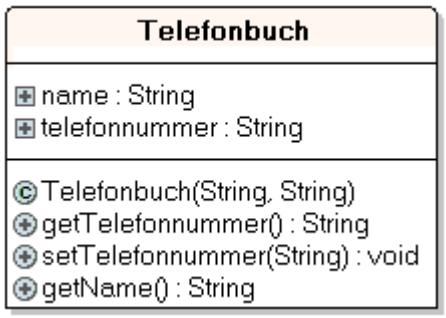


Abbildung 19: Klassendiagramm Telefonbuch

```
public class Telefonbuch {

  // Eigenschaften
  private String name;
  private String telefonnummer;

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

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

  public void setTelefonnummer(String telefonnummer) {
    this.telefonnummer = telefonnummer;
  }

  public String getName() {
    return name;
  }

  // Ende Ereignisprozeduren
}
```

2.6.3 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.

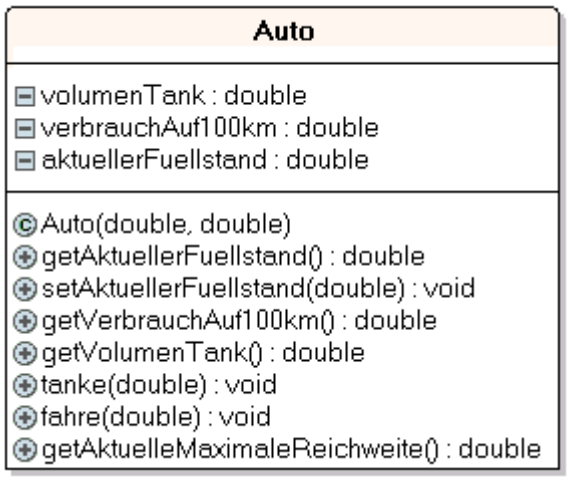


Abbildung 20: Klassendiagramm des Autos, bei dem nur die Tankdaten interessieren

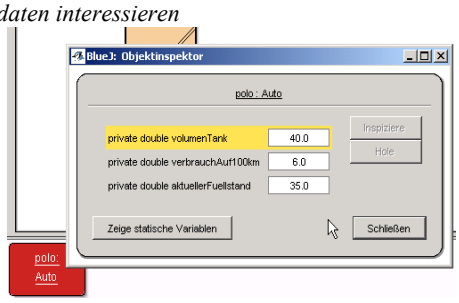


Abbildung 21: Die Daten meines 1990er Polos

```
public class Auto {

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

  // Anfang Ereignisprozeduren

  // Konstruktor
  public Auto(double volumenTank, double verbrauchAuf100km) {
```

```
475     this.volumenTank=volumenTank;
        this.verbrauchAuf100km=verbrauchAuf100km;
        this.aktuellerFuellstand=0; // Bei Auslieferung ist der Tank leer
      }

480   public double getAktuellerFuellstand() {
        return aktuellerFuellstand;
      }

      public void setAktuellerFuellstand(double aktuellerFuellstand) {
485     this.aktuellerFuellstand = aktuellerFuellstand;
      }

      public double getVerbrauchAuf100km() {
        return verbrauchAuf100km;
490   }

      public double getVolumenTank() {
        return volumenTank;
      }
495
      public void tanke(double menge) {
        this.aktuellerFuellstand+=menge; // fuege Menge hinzu
      }

500   public void fahre(double kilometerZahl) {
        // Die Formel sagt folgendes aus:
        // vom Fuellstand wird der Verbrauch bei der Kilometerzahl abgezogen
        this.aktuellerFuellstand -= kilometerZahl / 100 * verbrauchAuf100km;
      }
505
      public double getAktuelleMaximaleReichweite() {
        return this.aktuellerFuellstand / verbrauchAuf100km * 100;
      }

510   // 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 Zahl)
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 zaehler und nenner und realisieren Sie die Methoden addierenEinerGanzenZahl, kuerzen und erweitern.

Übung:	
	Wenn Sie schon Vorwissen haben, können Sie hier für eine erweiterte Variante schauen: http://www.u-helmich.de/inf/BlueJ/exkurse/bruchrechnung.html

2.7 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 betrag, Konto zielkonto)
{
   this.abheben(betrag);
   zielkonto.einzahhlen(betrag);
}
```

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).

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

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.

Diese beiden Zeilen übernehmen nun die Arbeit:

Die Methode des aktuellen Objektes spricht man mit **this.methodenname( )** an.
Die Methoden eines anderen Objektes spricht man mit folgender Konstruktion an: **objekt.methodenname( )**, dabei muss das Objekt entweder in der Klasse angelegt worden sein oder als Übergabeparameter einer Methode übergeben worden sein.

In BlueJ muss ich zur Demonstration der Methode überweisen zumindest 2 Konten anlegen (denn wenn ich von einem Konto auf sich selbst überweise, stelle ich keine Veränderung fest, obwohl das zulässig ist). Als Parameter zielkonto muss ich nun den Objektnamen eingeben oder, was einfacher ist, auf das Objektdiagramm des Kontos klicken, das das Geld bekommen soll (dann trägt BlueJ selbst den Objektnamen ein).

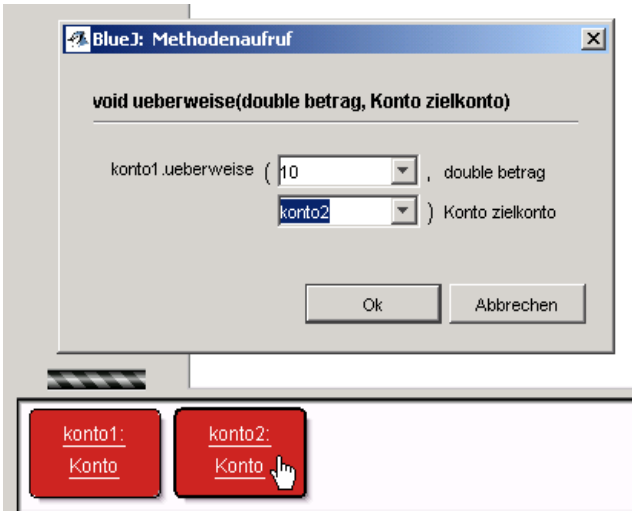


Abbildung 22: Ich klicke auf konto2, um den Namen als Parameter einzutragen

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.
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?

2.8 Anhang A – Überweisen in 3 Varianten

```
535
    public void ueberweise(double betrag, Konto zielkonto)
      {
         double zwischenspeicher;
         this.kontostand -= betrag;
540      // Um den Betrag zu buchen, schlage ich 3 Varianten vor.
         // Bitte entfernen Sie die Kommentierung, wenn Sie eine andere Variante
         // testen wollen.

         // *** Variante 1: ***
545      // 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:  ***
550      zwischenspeicher = zielkonto.getKontostand();
         zwischenspeicher += betrag;
         zielkonto.setKontostand(zwischenspeicher);
         // oder auch in kompakter Form ohne Zwischenspeicher
         // *** Variante 3: ***
555      // zielkonto.setKontostand(zielkonto.getKontostand()+betrag);
      }
```

2.9 Anhang B – Die Online-Bank Quelltext komplett

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

      //Konstruktor
      public Konto (String besitzerName){
565     this.besitzerName = besitzerName;
        this.kontostand = 0.0;
      }

      // Methoden
570   public void setBesitzerName(String neuerName)
      {
         this.besitzerName = neuerName;
      }

575   public String getBesitzerName()
      {
         return this.besitzerName;
      }

580   public void setKontostand(double betrag)
      {
         this.kontostand = betrag;
      }

585   public double getKontostand()
      {
         return this.kontostand;
      }
```

```
590   public void einzahlen(double betrag)
      {
         this.kontostand += betrag;
      }

595   public void ueberweise(double betrag, Konto zielkonto)
      {
         double zwischenspeicher;
         this.kontostand -= betrag;
         zielkonto.einzahlen(betrag);
600   }

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

2.10 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

3 Version

- 0.1 - 04.12.2005 - Pi - Klassen und Quelltext, Festlegungen
- 0.2 – 07.02.2006 – Pi – Objekte ausgebaut
- 0.3 – 9.2.2006 – Pi – überarbeitet, Beispiele und Überweisung
- 0.4 – 15.03.2006 – Pi – Konzeptioneller Fehler in Überweisen, Nummerierungen, neue Formatvorlage
- 0.5 – 28.03.2006 – Pi – Anhang: Onlinebank, this vereinheitlicht

Wikipedia
aus: Quelle

Tabelle Gelb

Übung:	
1)	
2)	
3)	

 Wikipedia

 Wikibook

 ZUM