Why BlueJ?
Java is quickly becoming very popular, both for application development and for teaching, and many programming environments exist. This raises the question: Why yet another programming environment?
The short answer is: because none of the existing environments is suitable for teaching.
In particular, we want an environment that is really object-oriented. An environment that emphasises classes and objects as its basic units of interaction, so that students are naturally led into a way of thinking in terms of classes and objects.
Secondly, we want an environment that is sophisticated[1], but at the same time easy to use, so that students can start using it straight away without the need for long introduction. We do not want to teach about the environment itself.
Thirdly, we want an environment that supports interaction and experimentation.
Fourthly, we want an environment that uses visualisation to display class structure. After all, half of the time in a teaching context we talk about class structure. Structuring the problem is one of the most important aspects of object-oriented programming. Yet many teachers find that students have difficulty thinking in terms of classes and objects. This is not surprising if all they ever see on the screen are lines of code or interface buttons and menus!
http://www.bluej.org/about/why.html

Einführung in die Objektorientierte Programmierung mit Java

Kapitel 2 – Java im Alltag

Inhaltsverzeichnis

"Der Computer arbeitet deshalb so schnell, weil er nicht denkt."
Gabriel Laub (1928-1998) *polnisch-tschechischer Schriftsteller und Satiriker*
http://de.wikiquote.org/wiki/Gabriel_Laub

1 sophisticated -> anspruchsvoll, ausgeklügelt

1 Java im Alltag

1.1. Konsolenausgabe

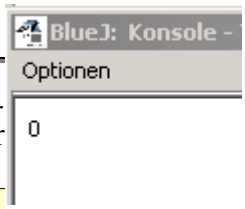


Abbildung 1: Konsolenausgabe

Der Befehl **System.out.println (...);** ist eine einfache Art der Ausgabe in Java. BlueJ öffnet eine „Konsolenanzeige“ (ein Fenster, in dem Ausgaben stehen), auf der das ausgegeben wird, was Sie in die Klammern geschrieben haben.

System.out.println (...); können Sie auf folgende Weise anwenden:

- **System.out.println (7+72/(3*12));** - zum Ausgeben einer Rechnung
- **System.out.println (“Hallo Welt“); -** zum Ausgeben eines Textes (Anführungsstriche!)
- **System.out.println (x); -** zum Ausgeben des Wertes einer Variable
- Das **Plus-Zeichen verbindet zwei Ausgaben**, wenn es sich nicht um eine reine Rechnung handelt. Der Fachausdurck für diese Operation ist „**Konkatenation“**
  System.out.println (“Der Wert von x ist “+x+“, verstanden?“);
- **System.out.println (x + 27);** - lässt den Wert der Rechnung ausgeben. **Wichtig: x wird hierbei nicht verändert!**
- **Mit der Zeichenkombination \n können Sie eine neue Zeile beginnen: “Hal\nlo“**

Übrigens ist println( ) eine Methode von System.out, d.h. Java wendet genau das an, was Sie im vorherigen Kapitel über Klassen gelernt haben. Die viele Befehle von Java sind Methoden von Klassen.

Wie kann ich im Java-Quelltext rechnen?
In Java dürfen auf verschiedene Weise Rechnungen durchgeführt werden. Nicht erlaubt ist, die Rechnung einfach so in einen Quelltext zu schreiben (z.B. 3*x;). Rechnungen sind in folgenden Zusammenhängen erlaubt:

- **Rechnungen bei Ausgaben**
  System.out.println (x * 2); // Hinweis: der Wert von x wird nicht geändert.
- **Rechnungen bei Zuweisungen**
  y = x * 2; // Hinweis: x wird wieder nicht geändert, aber y bekommt einen neuen Wert
- **Rechnungen bei Rückgaben**
  return (2 * x) // Hinweis: der Wert von x wird nicht verändert.
- **Rechnungen bei Parameterübergaben innerhalb von Methodenaufrufen**
  konto1.ueberweise(2*x); // Hinweis: der Wert von x wird nicht verändert. Verändert wird aber die Variable, die im Methodenquelltext steht: ueberweise(double betrag), dann bekommt betrag den Wert 2*x zugewiesen.
- **Rechnungen innerhalb von Vergleichen (siehe später: if und while)**
  if ((2*x) < 12) ... // Hinweis: der Wert von x wird nicht verändert.

(Faust-)Regel: Nur eine Zuweisung oder eine Parameterübergabe kann den Wert einer Variablen verändern[2]. Konsolenausgaben, Rückgaben etc. belassen die Werte der Variablen.

Übung	
1)	Schreiben Sie für das Konto eine Methode Kontoauszug, die die Eigenschaften des Kontos auf der Konsole (also mit System.out.println()) ausgibt.

2 Java bietet noch weitere Möglichkeiten, z.B. Zuweisungen in Vergleichen, auf die aber hier nicht eingegangen wird.

1.2. Elementare Datentypen in Java

Ein Datentyp (kurz Typ) besteht aus einem Wertebereich und darauf definierten Operationen. In der Programmierpraxis handelt es sich dabei um eine Festlegung, wie ein Speicherbereich vom Computer interpretiert werden soll.
vgl. http://de.wikipedia.org/wiki/Datentyp
Java bietet verschiedene **elementare Datentypen** an. Dabei handelt es sich um einige wichtige, eben elementare, Datentypen, die selbst keine Objekte sind. Ihnen werden hier nur 3 elementare Datentypen vorgestellt. Es wird für unsere Einsatzbereiche genügen, diese Datentypen zu kennen.
Die **Auswahl des Datentyps** fällt bei diesen dreien nicht schwer:

- Verwalte ich mit einer Eigenschaft oder einer Variable eine „entweder-oder"-Frage, die sich immer mit Ja oder Nein beantworten lässt, dann wähle ich **boolean** (Z.B. Geschlecht, istSchwanger, istEinLebewesen...[3])
- Benötige ich eine Zählvariable, bei der nur ganze Zahlen vorkommen, dann wähle ich **int**, es sei denn, 2.147.483.647 ist mir zu wenig. Im Gegensatz zu double benötigt int weniger Platz und Rechnungen lassen sich mit int schneller ausführen, da kein Nachkommaanteil benötigt wird. Rechenungenauigkeiten gibt es bei int nicht, solange man in dem gültigen Bereich bleibt (z.B. zählen von Anzahlen oder Wiederholungen, Alter in Jahren, ...).
- Benötige ich eine Zahl, die auch Nachkommastellen haben kann, so wähle ich **double**. In einem gewissen Maße muss ich mit Rechenungenauigkeiten rechnen, da double nur eingeschränkt genau ist. Ihnen werden Zahlen wie 3.0000000000001 begenen, wenn Sie mit double rechnen (besonders bei Division und Multiplikation).

1.2.1 Wichtige elementare Datentypen und deren Kennzeichen in Java in der Übersicht

Datentyp	Wertebereich	Speicher-bedarf in Java	wichtige Operationen
boolean	true / false	8 bit	! (logisches Nicht), && (logisches Und), || (logisches Oder), == (vergleichendes Gleich), != (ungleich)
int (nur ganze Zahlen)	-2.147.483.648 bis 2.147.483.647	32 bit	+, -, *, /, <, >, <=, >=, ==, != (ungleich)
double (auch Kommastellen)	+/-4,9E-324 bzw. +/-1,7E+308	64 bit	+, -, *, /, <, >, <=, >=, ==, != (ungleich)

Exkurs: Weitere elementare Datentypen, die wir aber nicht benutzen

```
char    16 bit  0 ... 65.535 (z. B. 'A')
byte    8 bit   -128 ... 127
short   16 bit  -32.768 ... 32.767
int     32 bit  -2.147.483.648 ... 2.147.483.647
long    64 bit  -9.223.372.036.854.775.808 ... 9.223.372.036.854.775.807
float   32 bit  +/-1,4E-45 ... +/-3,4E+38
double  64 bit  +/-4,9E-324 ... +/-1,7E+308
```

3 Vorsicht Falle: Wenn etwas nicht bekannt ist, bietet sich manchmal ein dritter Wert an: unbekannt. In diesem Fall ist boolean nicht geeignet. Möglich ist es, in diesem Fall int einzusetzen: 0 – Nein, 1 – Ja, -1 – Unbekannt.

Übung:	
1)	Welchen Datentypen würden Sie wählen: ● Simulation eines Thermometers ● „Was bin ich“-Quiz-Fragen ● TOP 10 Chartposition ● Preise im Supermarkt ● Personalausweisnummer ● Telefonnummer ● Datum
2)	Fügen Sie zur Klasse Konto eine Testmethode hinzu, um folgendes zu testen: ● Was macht Java, wenn der Wertebereich einer Variable verlassen wird? ● Was macht Java, wenn man eine int Variable so dividiert, dass keine ganze Zahl herauskommt? ● Was macht Java, wenn man durch 0 dividieren möchte?

1.3. Runden von Zahlen

Variablen vom Typen double haben mitunter sehr viele Nachkommastellen. Mit folgendem Trick schaffen wir es, die Variablen auf zwei Stellen zu runden.
Dazu benutzen wir die Klasse Math, in der Methoden zu mathematischen Rechnungen. Damit man sie benutzen kann, muss man ganz an den Anfang der Klasse hinzufügen:

```
import java.lang.Math;
```

Vorgehen: Eine Zahl, z.B. 3.14159265 multipliziere ich mit 100: 314.159265.
Dann runde ich und erhalte 314. Danach teile ich durch 100 und bekomme 3.14.

```
// Je nach Version wird ganz am Anfang folgende Zeile benötigt:
import java.lang.Math;

// (...)

public double rundeAufZweiStellen(double zahl)
{
      return Math.round( zahl * 100 ) / 100;
}
```

Übung:	
1)	Wie ruft man diese Methode auf, wenn man eine Zahl runden möchte?
2)	Schreiben Sie die Methode rundeAufDreiStellen(). Für Fortgeschrittene: RundeAufXStellen()
3)	Recherchieren Sie nach den weiteren Möglichkeiten der Klasse Math, indem Sie in eine Suchmaschine als Stichwort java.lang.Math eingeben. Sie können auch hier schauen: http://www.galileocomputing.de/openbook/javainsel4/javainsel_05_001.htm#Rxx365java05001040001881F025100
4)	Warum ergibt Math.sin(90) nicht 1 sondern 0.89?

Vgl. auch http://www.zum.de/wiki/index.php/Runden_in_Java

Exkurs: Arithmetik in Java siehe
http://www.galileocomputing.de/openbook/javainsel4/javainsel_05_000.htm#Xxx999378

1.4. Der Datentyp String

Sie werden sich sicherlich schon gewundert haben, weshalb der Datentyp String noch gar nicht wieder angesprochen wurde. Der Grund hierfür ist, dass ein String ein zusammengesetzter Datentyp ist und kein elementarer.
Technisch gesehen ist ein String eine Aneinanderreihung von Einzelbuchstaben (vgl. Datentyp char – ein einzelner Buchstabe, diesen Datentypen werden wir hier nicht näher behandeln).
String ist eine Klasse, deshalb wird String auch im Gegensatz zu int oder double groß geschieben.
Ein vereinfachtes Klassendiagramm von String könnte so aussehen:

String
(nicht bekannt, da private)
+ length() : int + equals(String x):boolean + charAt(x : int) ...

Klassendiagramm zu String

String hat viele Methoden, von denen Sie jetzt 3 kennen lernen sollen. Die folgende Klasse bietet eine zugegeben sehr unsaubere Möglichkeit, die Methoden von String zu testen.

```
// eine extrem unsauber geschriebene Klasse, die nur zu Testzwecken existiert!
public class Stringtest
{
        public Stringtest(String s){
            System.out.println(s);
            System.out.println(s.length());
            System.out.println(s.charAt(0));
            System.out.println(s.charAt(s.length()-1));
            System.out.println(s.equals("Hallo"));
        }
}
```

Erläuterung:

- s.**length()** ergibt die Anzahl der Zeichen von s.
- s.**charAt**(0) ergibt das erste Zeichen von String s. **Position 0 entspricht dem ersten Zeichen**.
  s.**charAt**(1) würde das zweite Zeichen ergeben usw.
- s.**charAt**(s.**length**()-1) ergibt das letzte Zeichen von s.
- **System.out.println**(s.**equals**("**Hallo**")); gibt ein true aus, wenn s dem String „Hallo“ entspricht, ansonsten wird false ausgegeben. Auf diese Weise lassen sich 2 String Variablen oder Eigenschaften miteinander vergleichen (z.B. bei der Online-Bank: Hat der Besitzer des Kontos schon ein Konto bei der Bank?).

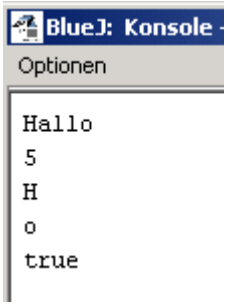


Abbildung 2: Konsolenausgabe nach Ausführung – s hatte den Wert “Hallo“.

Übung:	
1)	Warum wird das -1 bei s.**charAt**(s.**length**()-1) benötigt?
2)	Welche Länge hat ein leerer String? Lässt sich der erste Buchstabe eines leeren Strings ausgeben?
3)	Ein Wort mit 5 Buchstaben (z.B. „Hallo“), das von BlueJ als Parameter eingelesen wird, soll rückwärts in der Konsole erscheinen. Verändern Sie die Methode Stringtest entsprechend.

Übung:	
4)	Recherchieren Sie im Internet über die Möglichkeiten von Strings bei Java. Schauen Sie z.B. hier: ● http://java.sun.com/j2se/1.4.2/docs/api/java/lang/String.html (offizielle Java-Dokumentation der Firma Sun) ● http://www.galileocomputing.de/openbook/javainsel4/javainsel_04_000.htm#Rxx365java04000040001241F021100 (Kostenlos im Internet verfügbares Java-Buch, was für den Unterricht viel zu weit geht, das Sie sich aber anschauen sollten, falls Sie das Thema privat interessiert) ● http://de.wikibooks.org/wiki/Programmierkurs_Java:_Strings Hintergrundinformationen zu Strings als Wikibook.

1.5. Zur Umwandlung von Datentypen Typecasting

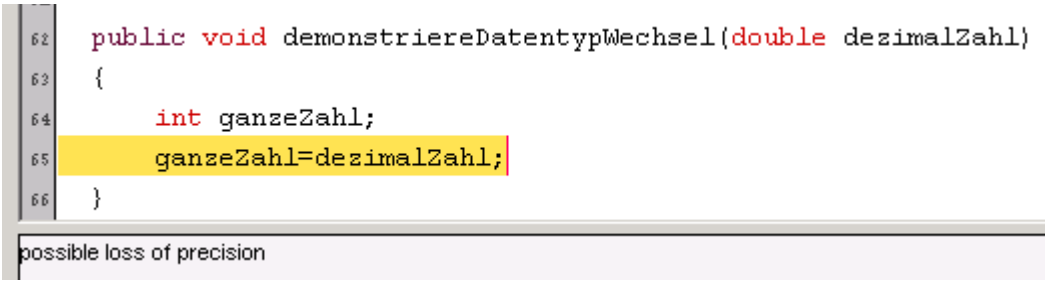


Abbildung 3: Die Umwandlung von double zu int ist geht nicht so einfach wie hier.

BlueJ weigert sich, eine Klasse zu übersetzen, die die oben abgebildetet Methode beinhaltet. Der Grund ist, dass bei einer Umwandlung von double in int Informationen verloren gehen könnten.

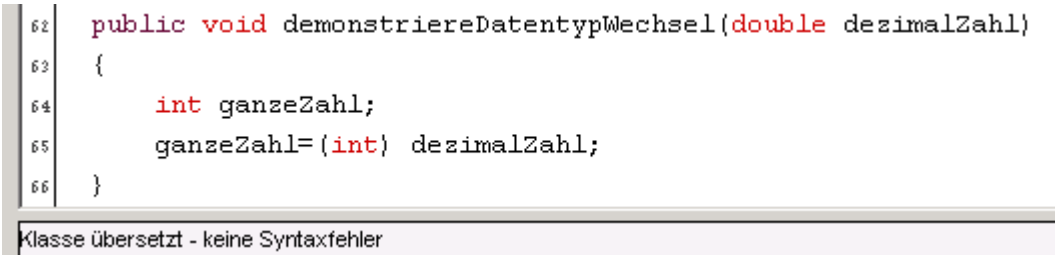


Abbildung 4: (int) löst das Problem

Die folgende Methode zeigt, wie man Datentypen konvertieren (Fachbegriff: Typecasting) kann. Für die Umwandlung einer Zahl in einen String gibt es den netten Trick, sie an einen leeren String zu hängen: “ “+213 ergibt den String “213“.

```
public void demonstriereDatentypWechsel(double dezimalZahl)
  {
      int ganzeZahl;
      ganzeZahl=(int) dezimalZahl;
      double dezimalZahl2;
      dezimalZahl2=(double) ganzeZahl;
      String eigentlichKeineZahl;
      eigentlichKeineZahl=""+dezimalZahl2;
      dezimalZahl2=Double.parseDouble(eigentlichKeineZahl);
      ganzeZahl=Integer.parseInt(eigentlichKeineZahl);
  }
```

Übung:	
1)	Wozu benötigt man die Umwandlung von Datentypen?
2)	Analysieren Sie die vorliegenden Methode und schreiben Sie heraus, wie man von welchen Datentypen in welchen Datentypen umwandeln kann.
3)	Wandeln Sie eine Stringeingabe zu einer Integerzahl um.

2 Kontrollstrukturen

Kontrollstrukturen (genauer *Steuerkonstrukte*) werden in Programmiersprachen verwendet, um den Ablauf eines Computerprogramms zu steuern. Eine Kontrollstruktur gehört entweder zur Gruppe der *Verzweigungen* oder der *Schleifen*. Meist wird ihre Ausführung über logische Ausdrücke der booleschen Algebra beeinflusst.
http://de.wikipedia.org/wiki/Kontrollstruktur

2.1. Bedingte Ausführung mit if

Ein wichtiges Element der Programmierung ist die bedingte Ausführung. Beispiele hierfür findet man allerdings auch in ganz alltäglichen Bereichen:

- Wenn man das Licht am Auto anlässt, dann ist die Batterie leer
- **if** (licht==“an“) {
  batterie = “leer“;
  }

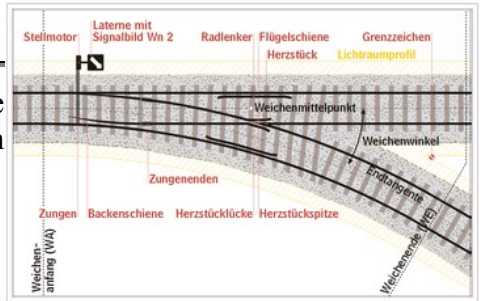


Abbildung 5: Eine Weiche

Mit **if** (*engl. falls, wenn*) lassen sich Programmabläufe abhängig von einer Bedingung steuern. Nur falls die Bedingung in den Klammern erfüllt ist, wird der nachfolgende Befehl ausgeführt.

Syntax	*Beispiel*

if (Bedingung**)** {
...
Anweisungsblock
...
}
	int x=12;
	if (x>=4) {
	System.out.println(“x ist größer als 3.“);
	}

Die **Bedingung ist** übrigens **immer eine boolsche Aussage**: x>=3 ergibt im Falle des Beispiels das Ergebnis true. Boolsche Variablen können sich also auch direkt abfragen:

Beispiel

boolean x=true;
if (x) {
System.out.println(“x ist true.“);
}

2.2. Wenn ... ansonsten – if ... else

Mit **if** (*engl. falls, wenn*) lassen sich Programmabläufe abhängig von einer Bedingung steuern. Nur falls die Bedingung in den Klammern erfüllt ist, wird der nachfolgende Befehl ausgeführt.

Syntax	*Beispiel*

if (Bedingung) {
Anweisungsblock 1
}
else {
Anweisungsblock 2
}
	int x=12;
	if (x>=4) {
	System.out.println(“x ist größer als 3.“);
	}
	else {
	Sytsem.out.println(“x ist nicht größer als 3.“);
	}

Ein konkretes Beispiel: Die Auszahlung erfolgt nur, wenn der es der Kontostand zulässt.

```
   public void auszahlen(double betrag)
   {
      if (kontostand>=betrag) {
170        kontostand-=betrag;
      }
      else {
         System.out.println("Auszahlung verweigert.\nZu wenig Geld!");
      }
175   System.out.println("Auf Wiedersehen!");
   }
```

Die folgende Methode prüft, ob der Kontostand positiv ist[4]. Typisch ist hierbei, dass Prüfmethoden wie diese mit dem Wort „is“, also dem englischen ist, anfangen. Der Rückgabewert der Methode ist vom Typen boolean, d.h. er wird entweder true oder false zurückgegeben.

180

Neu ist die Speicherungen von Daten in einer „lokalen Variable“ ergebnis. Diese ist eine lokale Variable, da sie nur in isPositiv gültig ist.

Lokale Variable: Eine Variable, die innerhalb einer Methode angelegt wird, wird lokale Variable genannt. Sie wird mit verlassen der Methode wieder zerstört. Sie kann nur innerhalb der Methode abgefragt und verändert werden.
Nennt man eine lokale Variable genauso wie eine Eigenschaft der Klasse, so muss man zur Unterscheidung das Wort this verwenden, um die Eigenschaft zu erreichen. Bei Namensgleichheit hat das Abfragen der lokalen Variable Priorität.
Lokale Variablen lassen sich gut als kurzfristigen **Zwischenspeicher** einsetzen.
Konvention: Die Deklaration der lokalen Variable sollte direkt unter dem Kopf der Methode – also direkt nach der ersten geschweiften Klammer - passieren.

```
185   public boolean isPositiv()
   {
      boolean ergebnis;
      if (kontostand>0) {
         ergebnis=true;
190   }
      else {
         ergebnis=false;
      }
      return ergebnis;
195   }
```

4 Eigentlich ist 0 sowohl positiv als auch negativ. Das berücksichtigt die Methode aber nicht. Ggf. müsste man die Methode isGroesserNull() nennen.

Übung:	
1)	Ändern Sie die Methode überweisen() so, dass auch hier geprüft wird, ob noch genügend Geld vorhanden ist.
2)	Fügen Sie eine Eigenschaft Dispo-Kredit mit einer get- und set-Methode ein. Ändern Sie anschließend die Methoden abheben() und überweisen() so, dass der Dispo-Kredit nicht überschritten werden kann. Schreiben Sie die Methode: isImDispo(), die true zurückgibt, wenn der Kontostand kleiner 0 ist.
3)	Durch Einzahlen eines negativen Betrages kann man den Bankautomaten austricksen. Ändern Sie diese Sicherheitslücke (umgehend ;-)).
4)	Erweitern Sie das Konto um eine Eigenschaft „gesperrt“. Implementieren Sie die zugehörige Methode isGesperrt(), die true zurückgibt, wenn das Konto gesperrt ist.

2.3. Prüfen von Programmen: Der Schreibtischtest

Der **Schreibtischtest** ist ein Verfahren, dass im Bereich der Softwareentwicklung verwendet wird, um Algorithmen oder Routinen auf Richtigkeit zu prüfen.
Der Schreibtischtest wird nicht mit Hilfe eines Rechners durchgeführt sondern vielmehr im Kopf bzw. auf dem Schreibtisch des Entwicklers. Dazu werden für einen Programmablauf eine Eingabemenge festgelegt. Anschließend wird mit jedem Element der Eingabemenge durch schrittweises Durchrechnen die Korrektheit des Programmablaufs überprüft.
nach http://de.wikipedia.org/wiki/Schreibtischtest

Die bislang noch sehr einfachen Beispiele eignen sich gut, um den Schreibtischtest zu demosntrieren. Nehmen wir die Methode auszahlen

```
1  public void auszahlen(double betrag)
2  {
3     if (kontostand>=betrag) {
4         kontostand-=betrag;
5     }
6     else {
7         System.out.println("Auszahlung verweigert.\nZu wenig Geld!");
8     }
9     System.out.println("Auf Wiedersehen!");
10 }
```

Zunächst notiere ich **alle Parameter, Eigenschaften, Variablen, Bedingungsabfragen, Rück- und Ausgaben**, die in der Methode vorkommen. Die erste Spalte nehme ich für die Zeilenkennzeichnung. In diesem Fall ergibt sich folgende Tabelle:

Zeile	**betrag**	**kontostand**	**kontostand >= betrag**	**Ausgabe**

Nun benötige ich, damit ich loslegen kann, noch **Ausgangswerte**. Diese stehen **entweder in der Aufgabenstellung oder ich muss mir sinnolle Ausgangswerte ausdenken**, die den Programmablauf angemessen darlegen. In diesem Fall möchte ich erreichen, dass einmal a) der if- und einmal b) der else-Zweig ausgeführt werden.
a) Ich wähle zunächst für betrag 100 und für kontostand 200 und trage das ein. Nun schaue ich, wann sich bei meinen Spalten etwas ändert. Jede Änderung schreibe ich in eine neue Zeile.

Zeile	betrag	kontostand	kontostand >= betrag	Ausgabe
	50	27		
3			false	
4		100		
9				„Auf Wiedersehen“

Erläuterung: Da der Kontostand hoch genug ist (kontostand >= betrag ist true), werden 100 Euro vom Kontostand abgezogen. Der else-Zweig wird übersprungen. Die Ausgabe in Zeile 9 erfolgt aber.

b) Was passiert, wenn ich zu wenig Geld auf dem Konto habe.

Zeile	betrag	kontostand	kontostand >= betrag	Ausgabe
	100	200		
3			false	
7				„Auszahlung verweigert ...“
9				„Auf Wiedersehen“

Erläuterung: Da der Kontostand nicht hoch genug ist (kontostand >= betrag ist false), wird der else-Zweig ausgeführt. Es wird kein Geld abgezogen. Statt dessen passiert eine Ausgabe in Zeile 7. Die Ausgabe in Zeile 9 erfolgt.

Übung:	
1)	Erläutern Sie die Bedeutung eines Schreibtischtestes in Bezug auf Verständnis des Programmablaufs und Fehlersuche. Schreiben Sie eine Schritt für Schritt Anleitung, die für jeden Schreibtischtest gilt.
2)	...

2.4. Logische Operatoren NICHT, UND und ODER

Wichtige logische Operatoen sind NICHT, UND und ODER. Wir verwenden sie alltäglich, wenn wir Aussagen formulieren. Allerdings sind wir im Alltag häufig unpräzise. Im Alltag verwenden wir Logische Operatioren ohne es zu bemerken häufig bei Sätzen mit „Wenn...“:

- Wenn ich eine gute Note in der Klausur habe UND ich im Unterricht mitmache, dann bekomme ich eine gute Note (d.h. nur wenn beides der Fall ist).
  In Java: if ((klausurNote<=2) && (soMiNote <=2)) ...
- Wenn es regnet ODER die Sonne stark scheint, sitze ich unter einem Schirm (d.h. wenn **zumindest eines erfüllt** ist. Wenn es regnet und zugleich die Sonne stark scheint, sitze ich natürlich erst recht unter einem Schirm.)
  In Java: if ((sonneScheint==true) || (esRegnet==true)) ...
- Wenn es NICHT hell ist, dann geht die Straßenlaterne an.
  In Java: if (!(helligkeitsSensor>30)) ... ist das gleiche wie: if (helligkeitsSensor<=30) ...

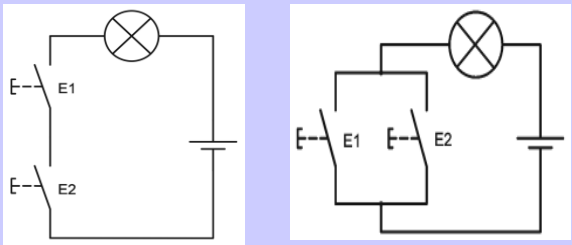
2.4.1 Alltagssprache und Logik: Missverständnisse

Eine andere Bedeutung hat das „aufzählende Und“ („eine Tasche und ein Fenster“). Dies hat nichts mit Logik zu tun. D.h. nicht jedes UND ist logisch.
Ungenau wird das ODER im Alltag verwendet: „Möchtest du ein Eis oder ein Bonbon“? Hier

kommt dem quängelnden Kind die Logik zu Gute: Bei einem logischen ODER dürfte es dann auch ein Eis und ein Bonbon auswählen. „Ein Eis oder ein Bonbon“ also genau gesagt „ein entweder oder“ entspricht dem logischen „exklusiven Oder“ (XOR), das wir hier nicht näher thematisieren.

Zusammenfassung: Für zwei boolsche Werte (also bei Java Typ boolean) a und b gilt:

- **!a** (bedeutet NICHT a) erzeugt das Gegenteil von a. Wenn a true ist, ist !a false und andersherum.
- Die Aussage **a && b** (bedeutet a UND b) ist true, wenn **sowohl a als auch b** true sind. Sonst nicht.
- Die Aussage **a || b** ist true, wenn **entweder a oder b oder beide** true sind. D.h. nur wenn sowohl a als auch b false sind, ist die Aussage a || b auch false.

Übung:		
1)	Nennen Sie weitere Beispiele, wo logische Aussagen verwendet werden und prüfen Sie, ob es mit der Logik übereinstimmt.	
2)	Prüfen Sie die Richtigkeit der folgende Aussagen: Wenn (a || b) false ist, weiß ich, dass mindestens eine der Variablen true ist. Wenn (a && b) true ist, weiß ich, dass jede der Variablen true ist. Wenn !(a && b) false ist, weiß ich, dass keine der Variablen true ist. Wenn !(a || b) false ist, weiß ich, dass beide Variablen true sind. (a && b) || ! (a && b) ist immer true.	
3)	 aus Wikipedia (Die genaue Quellenangabe resultiert aus Lösung der Aufgabe und wird deshalb nicht genannt)	Die Schaltungsskizzen links charakterisieren logische Operatoren. Erläutern Sie den Zusammenhang. Vertiefung: Schlagen Sie in Wikipedia nach: ● Aussagenlogik ● Negation (Komplement-Gatter) ● Konjunktion (Und-Gatter) ● Disjunktion (Oder-Gatter)

Eine **Wahrheitstabelle** (auch Wahrheitstafel) ist eine Tabelle, die in der Aussagenlogik jeder Kombination einer bestimmten endlichen Anzahl von Wahrheitswerten einen bestimmten Resultatwert zuordnet. Sie wird genutzt um Wahrheitswertefunktionen bzw. boolesche Funktionen zu definieren oder darzustellen und um einfache aussagenlogische Beweise zu führen.

Die einfachste Wahrheitstabelle zeigt als ein Beispiel für eine einstellige Wahrheitswertfunktion einer zweiwertigen Logik das Ergebnis der Negation einer Aussage in der klassischen Aussagenlogik:

a	**NICHT bzw. !a (Negation)**
w	f
f	w

*Abbildung 6: Die **Straßenlaterne** kann als **Beispiel für eine Negation** gesehen werden: Ist es um sie herum hell, ist sie selbst nicht hell und anders herum. Bild aus: http://commons.wikimedia.org/wiki/Image:Praha_laterne-00-2003_12_22.jpg*

Die folgende Tabelle gibt für jeden Wahrheitswert der Aussagen a und b das Resultat einiger zweiwertiger Verknüpfungen an:

a	b	UND bzw. a && b (Konjunktion)	ODER bzw. a || b (Disjunktion)	Komplexeres Beispiel (a&&b) || a
f	f	f	f	f
f	w	f	w	f
w	f	f	w	w
w	w	w	w	w

http://de.wikipedia.org/wiki/Wahrheitstabelle

2.5. Anwendung: Die kleinste aus 3 Zahlen.

Drei Zahlen werden als Parameter übergeben, die kleinste soll zurückgegeben werden. Die folgende Methode liefert einen Lösungsansatz, der noch nicht vollkommen optimal ist (vgl. Aufgabe 2).

```
public int kleinsteZahl(int x, int y, int z)
{
    int zwischenspeicher=x;
    if ((x<y) && (x<z)) zwischenspeicher = x;
    if ((y<x) && (y<z)) zwischenspeicher = y;
    if ((z<x) && (z<y)) zwischenspeicher = z;
  return zwischenspeicher;
}
```

Übung:	
1)	Analysieren Sie die Methode und erläutern Sie dabei die Anwendung des logischen Unds.
2)	Wir reagiert die Methode auf die Eingabe von zwei gleich großen Zahlen y und z und einem größeren x. Prüfen Sie dies mit Hilfe eines Schreibtischtestes. Schreiben Sie die Methode so um, dass auch bei gleich großen Zahlen zumindest eine der kleinsten Zahlen ausgegeben wird.
3)	Realisieren Sie eine Variante für 4 Zahlen. Schätzen Sie ab, wie aufwändig eine Variante mit 5, 6, 7... Zahlen ist.
4)	Ingenieur Meyer hat einen Bankautomaten konzipiert, bei dem er sich ein Hintertürchen offen gelassen hat: Variante a) Eine Auszahlung ist möglich, wenn der Kontobesitzer Meyer heißt, auch wenn der Betrag nicht mehr verfügbar ist. Aus Angst, aufzufliegen, da es zu viele Meyers gibt, hat er folgende Variante realisiert: Variante b) Eine Auszahlung ist möglich, wenn der Kontobesitzer Meyer aber auch nur genau dann, wenn der Abhebebetrag 650 Euro ist, egal ob der Betrag verfügbar ist. Ergänzen Sie beide Varianten in Ihrer Onlinebank und diskutieren Sie über den Realismus dieser Aufgabe. Hinweis: Denken Sie daran, dass Sie die Abfrage des Namens mit equals angehen (siehe Kapitel String)!

2.5.1 Weitere Übungen

Übung:	
1)	Bestimmen Sie die Wahrheitstabelle zu a) ! (x || y) b) (x || y) && !y c) (x && y) || !(x && y)
2)	Im Freibad gilt folgende Regelung: Kinder unter 3 Jahren sind kostenlos. Kinder bis 12 kosten die Hälfte des Erwachsenenpreises. Ein Erwachsenener zahlt 5 Euro, es sei denn er ist über 65, dann zahlt er nur die Hälfte. Schreiben Sie eine Klasse Freibadkasse, die eine Methode hat die nach Übegabe des Alters entsprechend den Preis nennt. Eine zweite Methode soll nach Übergabe des Alters ein Ticket drucken und den Eintrittspreis auf die Eigenschaft tageseinnahmen addieren.
3)	Eine Klasse Kühlhaus hat 4 Eigenschaften: temperaturRaum1, temperaturRaum2, temperaturRaum3, kühlstufe Sobald in einem Raum die Temperatur größer als -5 Grad ist, schaltet kühlstufe auf Stufe 1. Liegt in mindestens 2 Räumen die Temperatur über -5 Grad, so wird Stufe 2 eingestellt. Realisieren Sie eine entsprechende Klasse mit den zugehörigen Methoden.
4)	Ein Computerhändler gibt folgende Rabatte: - bei Beträgen unter 500€ gibt es 5% Rabatt - bei Beträgen zwischen 500€ und 2000€ gibt er 10% Rabatt - über 2000€ gibt er sogar 15% Rabatt. Schreiben Sie die Methode: *druckeRechnungAus(double rechnungsbetrag)*
5)	*sortiereNachLaenge(String x, String y, String z)* soll auf der Konsole zunächst den kürzesten, dann den mittellangen und schließlich den längsten der drei Parameter ausgeben (Hinweis: vgl. Datentyp String: length)

2.6. Die Wiederholungsanweisung mit while

Versteckspiel: Als Suchender muss man zunächst bis 100 zählen, dann laut „Ich komme...“ rufen. Erst dann darf man zum Suchen aufbrechen .

Dieses simple Spiel ist ein einfacher Algorithmus:

- Die erste Zahl ist 1.
- Prüfe, ob die 100 erreicht wurde
- Wenn das noch nicht erfüllt ist, sage die Zahl und erhöhe die Zahl um eins.
- Wenn doch, dann rufe „Ich komme“

Neu daran ist die Wiederholung.

Mit **while** (*engl. solange*) lassen sich Programmabläufe abhängig von einer Bedingung wiederholen. Die Bedingung wird **vor** Ausführung der Anweisung abgefragt ("*prechecked loop*", "*kopfgesteuerte Schleife*").

Die Befehlsstruktur ist: **Solange <Bedingung erfüllt> führe <Anweisung> aus.**

Java-Syntax:
```
while (Bedingung)
{
    Anweisungsblock (was soll wiederholt werden?)
}
```

Wichtig:
Der Anwendungsblock muss so aufgebaut sein, dass die Bedingung irgendwann erfüllt ist, ansonsten bekommen Sie eine Endlosschleife.

2.6.1 Die Endlosschleife

Eine Schleife, die (eigentlich) niemals endet und dabei den Computer völlig auslastet ist eine typische Ursache für Abstürze.
Die folgende Methode ist eine Endlosschleife, die unendlich lange ausgeführt würde, wenn niemand die Ausführung stoppen würde. Typischer Fehler: Es wird x als Bedingung abgefragt: (x<100) , aber es wird x nicht innerhalb der Schleife verändert.

```
public void führeEndlosschleifeAus(){
  int x; // Eine Zählvariable x
      x=1;  // Anfangswert für x
  while (x <= 100) {
      System.out.println (x);        // schreibe x auf die Konsole
      // Typischer Fehler: x wird in der Schleife nicht erhöht.
  }
}
```

Bevor Sie die Methode ausprobieren, lesen Sie zunächst, wie man in BlueJ Programme stoppt:

2.6.2 Notunterbrechungen bei BlueJ

Mit **Shift+Umschalt+R** setzen Sie die Virtuelle Maschine zurück, was eine Beendigung der aktuell ausgeührten Methode bewirkt. Sie können auch auf der rot-weißen Ausführungsnanzeige mit der rechten Maustaste das Kontextmenu hervorholen und die Ausführung damit stoppen.

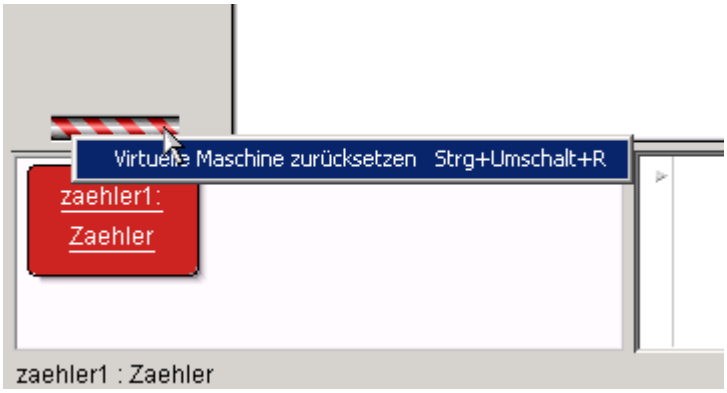


Abbildung 7: So unterbrechen Sie Methoden in BlueJ

2.6.3 Die Versteckspiel Zählung als Java-Quelltext

So sähe das Verstecken-Beispiel als Methode der Klasse Verstecken aus. Der Computer schreibt die Zahlen von 1 bis zahl in die Konsole.
Hinweis: Wenn die oberen Zahlen nicht mehr zu sehen sind, liegt das an der Konsole, die nur begrenzt viele Zeilen zulässt. Das Problem kann man umgehen, indem man statt println ein print einsetzt. Dann wird nicht bei jeder Zahl eine neue Zeile begonnen.

```
public class Verstecken
{
      public void zaehleBisZahlundRufe(int zahl)
      {
        int x;                              // Eine Zählvariable x
        x=1;                                // Anfangswert für x
        while (x <= zahl) {
            System.out.println (x);         // schreibe x auf die Konsole
            x++;                            // erhöhe x um 1
        }
        System.out.println ("Ich komme...!");
      }
}
```

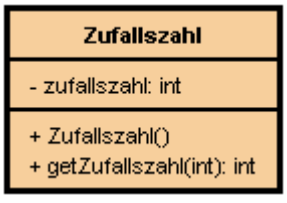
Hier findet sich keine Endlosschleife. x wird bei jeder Ausführung um 1 erhöht. Wenn x den Wert von Zahl überschritten hat, wird die Ausführung beendet.

Übung:	
1)	Analysieren Sie die Methode zaehleBisZahlundRufe **(int** zahl**)**, indem Sie folgende Dinge zunächst durchdenken und dann testen: a) Was passiert, wenn Sie beim Methodenaufruf 1 als zahl übergeben. b) Was passiert, wenn Sie beim Methodenaufruf negative Zahlen eingeben. c) Was verändert sich, wenn die Zeile mit x++ über der System.out Zeile steht? d) Wie können Sie in Zweierschritten zählen? e) Warum wird „Ich komme“ nur ein mal ausgegeben?
2)	Verändern Sie die Methode so, dass ein Coutntdown heruntergezählt wird.
3)	Lassen Sie neben zahl auch die Variable schwrittweite übergeben. Sie soll bestimmen, in wie großen Schritten heraufgezählt werden soll (z.B. Schrittwete 3 ergibt: 1, 4, 7, 10...

2.7. Übungen mit Zufallszahlen

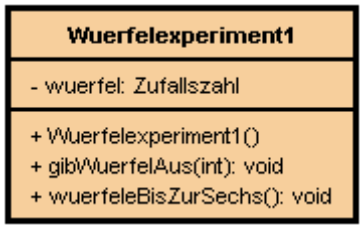
Übungen zu Zufallszahlen eignen sich, um die Interaktion zwischen verschiedenen Klassen zu demonstrieren. Die recht einfache Klasse Zufallszahl sieht wie folgt aus:

```
public class Zufallszahl
{
        private int zufallszahl;

        // gibt eine Zufallszahl zwischen 1 und maximum zurück
        public int getZufallszahl(int maximum)
        {
                return (int) Math.random()*maximum+1;
        }
}
```



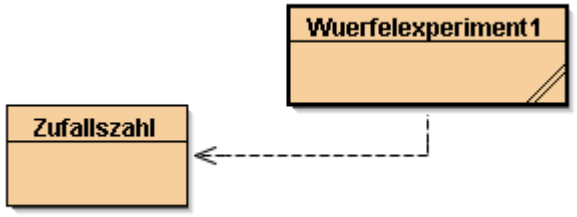
Diese Klasse kann man nun auf einfache Weise in anderen Klassen

verwenden. Eine Eigenschaft z.B. wuerfel vom Typ Zufallszahl wird vorbereitet (vgl. erste Zeilen). Nun lassen sich mit wuerfel.**getZufallszahl**(6) Zufallszahlen zwischen 1 und 6 bestimmen, die man z.B. ausgeben kann (erste Methode) oder einer Variable zuweisen kann (zweite Methode).




```
370 public class Wuerfelexperiment1
    {

        private Zufallszahl wuerfel;

375     public Wuerfelexperiment1 (){
            wuerfel=new Zufallszahl();
        }

        public void gibWuerfelAus(int anzahl)
380     {
            int i=1;
            while (i<=anzahl){
                System.out.println("Wurf "+i+ " ist "+wuerfel.getZufallszahl(6));
                i++;
385         }
        }

        public void wuerfeleBisZurSechs()
        {
390         int i=1;

            // Erster Wurf muss auf jeden Fall durchgeführt werden
            int zwischenspeicher=wuerfel.getZufallszahl(6);
            System.out.println("Wurf "+i+ " ist "+zwischenspeicher);
395
            // weitere Würfe
            while (zwischenspeicher!=6){
                i++;
                zwischenspeicher=wuerfel.getZufallszahl(6);
400             System.out.println("Wurf "+i+ " ist "+zwischenspeicher);
            }
        }

    }
```

405 Zufallszahl und Wuerfelexperiment1 haben eine „Assoziation“ (entspricht Beziehung) zueinander, sie assoziieren. Das drückt BlueJ (UML-konform) durch einen Pfeil aus.
In diesem Fall handelt es sich um eine gerichtete Assoziation. Zufallszahl wird von Wuerfelexperiment1 verwendet, andersherum aber nicht. „Navigieren“ von Würfelelement1 zu Zufallszahl ist erlaubt, andersherum nicht.



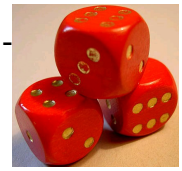
410 Vertiefend: http://de.wikipedia.org/wiki/Assoziation_(UML)

Übung: zu gibWuerfelAus(...)	
1)	Der Benutzer soll bei der Methode gibWuerfelAus selbst angeben können, wie viele Seiten der Würfel hat (es gibt z.B. auch 8-seitige oder 24-seitige Würfel).
2)	Zusätzlich soll die Summe der Würfe ausgegeben werden.
3)	Bestimmen Sie den Durchschnitt aller Würfe.
4)	Immer, wenn der vorherige Wurf gleich groß war, wie der aktuelle, soll „Gleich“ auf dem Bildschirm ausgegeben werden.
5)	Realisieren Sie eine Methode, die prüft, ob bei vielen Würfen jede Zahl annähernd gleich oft fällt (Kontrolle des Pseudozufallszahlgenerators).

Übung: Würfelexperiment2 – diesmal mit 2 Würfeln	
1)	Legen Sie einen zweiten Würfel an und schreiben Sie wie oben gibWuerfelAus(...).
2)	Lassen Sie immer zuerst den größeren Würfel ausgeben.
3)	Lassen Sie anzeigen, wenn ein Pasch gewürfelt wurde.
4)	Immer, wenn der vorherige Wurf von der Summe her gleich groß war, wie der aktuelle, soll „Gleich“ auf dem Bildschirm ausgegeben werden.

2.7.1 Aus der Geschichte des Würfelspiels

In der römischen Antike waren Würfelspiele in allen Schichten verbreitet, obwohl die Autoritäten sie mit Strafe bedrohten.
Von der Würfelleidenschaft der Germanen berichtet Tacitus in seiner *Germania*. Nach Tacitus spielten sie in nüchternem Zustand mit äußerstem Leichtsinn um Haus und Hof, zuletzt gar um die eigene Freiheit.
Der Zeitvertreib mit Suchtgefahr wurde auch im Mittelalter oft verboten, etwa im Jahr 1396 in Mailand. Ein Zuwiderhandelnder musste hier mit 200 Lire Bußgeld rechnen und sich danach mindestens 100 Meilen von der Stadt entfernen.
In englischen Spielsälen gab es um das Jahr 1800 menschliche "Würfelschlucker", deren Aufgabe es war, bei Razzien alle Würfel rasch hinunterzuschlingen.



http://de.wikipedia.org/wiki/W%C3%BCrfelspiel , http://de.wikipedia.org/wiki/Bild:Wuerfel_rot.jpg

2.7.2 Hintergrundwissen Zufallszahlen

Zur Erzeugung von Zufallszahlen gibt es verschiedene Verfahren. Diese werden als Zufallszahlengeneratoren bezeichnet. Ein entscheidendes Kriterium für Zufallszahlen ist, ob das Ergebnis der Generierung als unabhängig von früheren Ergebnissen angesehen werden kann oder nicht.
Echte Zufallszahlen werden mit Hilfe physikalischer Phänomene erzeugt: Münzwurf, Würfel, Roulette, Rauschen elektronischer Bauelemente, radioaktive Zerfallsprozesse oder quantenphysikalische Effekte. Diese Verfahren nennen sich physikalische Zufallszahlengeneratoren, sind jedoch zeitlich oder technisch recht aufwändig.

In der realen Anwendung genügt häufig eine Folge von Pseudozufallszahlen, das sind *scheinbar* zufällige Zahlen, die nach einem festen, reproduzierbaren Verfahren erzeugt werden.

http://de.wikipedia.org/wiki/Zufallszahl

2.8. 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 – 14.03.2006 – Pi – überarbeitet: Logische Fkt und weitere Aufgaben, Direkteingabe herausgenommen

Wikipedia
aus: Quelle

Tabelle Gelb

Übung:	
1)	
2)	
3)	

```
1
```

Wikipedia

Wikibook

ZUM