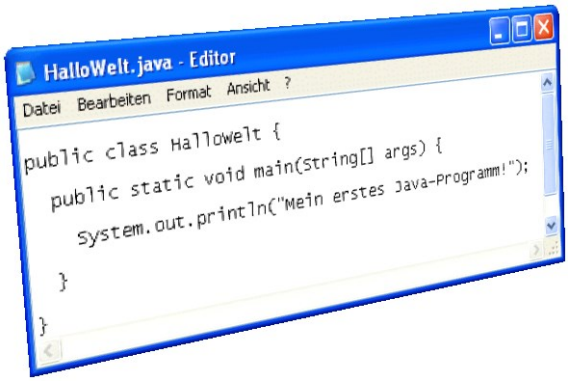
Einführung in das Programmieren (mit Java)

© Robert Wiedermann, http://www.rali.boku.ac.at/robertwiedermann.html Version: 2011

In den kommenden Übungsstunden möchten wir Ihnen einen **Einstieg in das Programmieren** bieten.



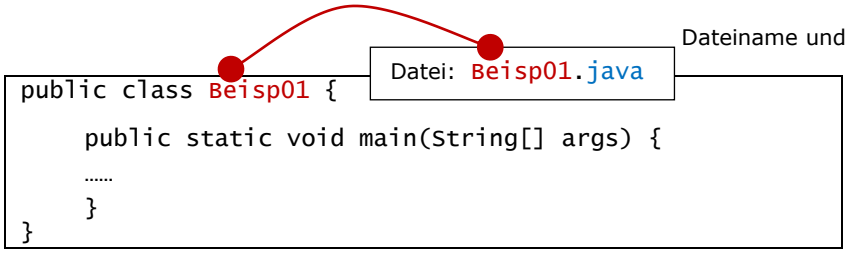
Programmiersprachen gibt es viele, und jede hat ihre Vor- und Nachteile. In Abstimmung mit den Bioinformatik-Lektoren der höheren Semester fiel für diesen Kurs die Wahl auf **Java**, und zwar aus folgenden Gründen:

- Java ist weit verbreitet und für die verschiedensten Betriebssysteme (Windows, Linux, Mac OS X) verfügbar.
- Die Syntax von Java ist der anderer sehr weit verbreiteter Sprachen wie C oder PHP sehr ähnlich, so dass Sie bei einer Einführung in Java einen guten Grundstock erhalten.
- Java steht kostenlos zur Verfügung, das Java Development Kit (JDK) kann frei herunter geladen werden: **http://java.sun.com/**
- Java und Perl sind die wichtigsten Sprachen in der **Bioinformatik**, es gibt auch spezielle Bioinformatik-Erweiterungen wie BioJava.
- Zu Java findet man in jeder Buchhandlung meterweise Literatur in jeder Gewichtsklasse. Falls Ihnen die Entscheidung schwer fällt, hier einige Empfehlungen von Büchern, die auch online zur Verfügung stehen:
  - Christian ULLENBOOM, "Java ist auch eine Insel": **http://openbook.galileocomputing.de/javainsel/** bzw. **http://www.tutego.de/javabuch/**
  - Guido KRÜGER, "Handbuch der Java-Programmierung": **http://www.javabuch.de/**
  - Albrecht WEINERT, "Java für Ingenieure": **http://www.a-weinert.de/java4ing/**

Dennoch versteht sich dieser Übungsteil *nicht* als Java-Spezialkurs, sondern als ein **allgemeiner Einstieg in die Programmierun**g, so dass auf Spezialitäten aus Gründen der Übersichtlichkeit möglichst verzichtet wird.

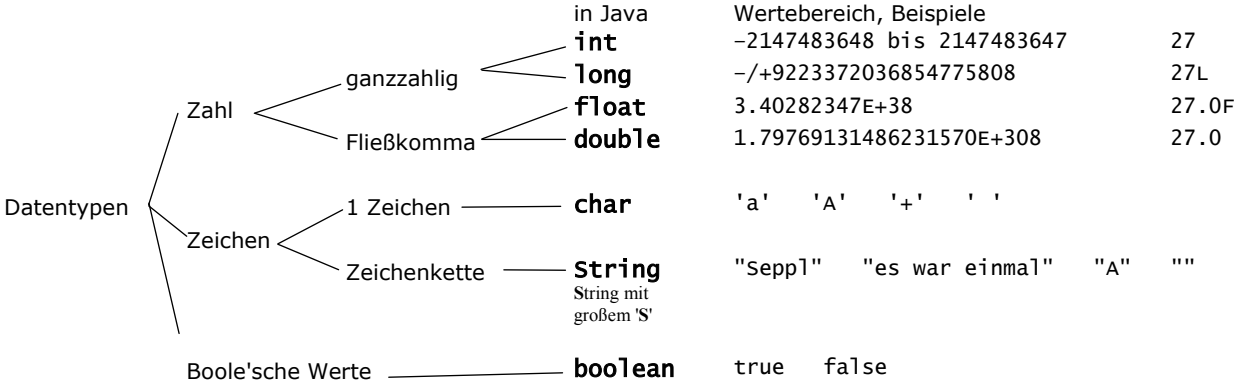
Programmieren in Java – das Wichtigste auf einem Blatt

Grundgerüst einer Java-Applikation



Dateiname und Klassenname müssen übereinstimmen!

Datentypen: (hier nur ein Auszug, Details siehe Literatur!)



Literale: `2 1.5 365L 2.5F 1.23E17 'b' '5' ' ' "es war einmal" "" true`

Operatoren: Arithmetische Operatoren (+, -, *, /), Zuweisungsoperator (=), Vergleichsoperatoren, …

Ausdrücke: `7-3 (5.23 + 11.8)/3.2 "Max"+" "+"Mustermann" 17 > 23`

Variable: eine Variable hat:

- einen **Namen** (Bezeichner, identifier):
  Empfehlung: Buchstaben a-z, A-Z, Ziffern 0-9 (nicht am Anfang), Unterstreichungszeichen (nicht am Anfang), keine Sonderzeichen (äöüß+-/*….), keinesfalls Leerzeichen; **case-sensitive**!
- einen **Datentyp** (siehe oben)

Variablendeklaration: Java ist eine streng typisierte Sprache, vor der Verwendung einer Variablen muss ihr Datentyp festgelegt werden –

```
int anzahl;        float preis, umrechnungskurs;
```

Wertzuweisung an Variable: rechts vom Zuweisungsoperator (=) steht ein Ausdruck, der ausgewertet, und dann der Variablen zugewiesen wird

```
maximum=5;      ergebnis=27.5*3;     vorname="Max";     mittelw=summe/anzahl;
```

Ausgabe:

```
System.out.print("Das Ergebnis lautet: ");    nachfolgende Ausgabe in der selben Zeile
System.out.println("Hallo Leute!");     print line macht einen Zeilenvorschub nach der Ausgabe
```

Kommentare: Falls Sie selbst oder jemand nach Ihnen Ihr Programm verstehen will/muss, sollten Sie nicht mit ausführlichen Erläuterungen in Form von Dokumentation und Kommentaren sparen. (Der Compiler sieht sich das nicht an, das ist nur für menschliche Leser gedacht.)

```
// einzeiliger Kommentar
/* mehrzeiliger Kommentar */
```

Verzweigung, Entscheidung:

In der runden Klammer steht eine Bedingung, also ein Ausdruck, der einen boole'schen Wert liefert

```
if ( Bedingung ) { Anweisungsblock } else { Anweisungsblock }
```

Schleifen, Wiederholung:

```
for ( int i = 1; i < 10; i=i+1 ) { Anweisungsblock, der wiederholt wird }
while ( Bedingung ) { Anweisungsblock, der wiederholt wird }
do { Anweisungsblock, der wiederholt wird } while ( Bedingung )
```

```
if ( Bedingung )
    {
    Anweisungsblock;
    }
else
    {
    Anweisungsblock;
    }

while ( Bedingung )
    {
    Anweisungsblock;
    }
```

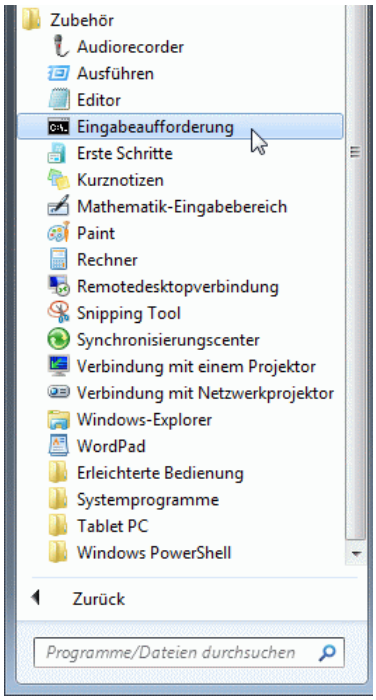
Array (indizierte Variable) besteht aus durchnummerierten Elementen

```
String[] wert = new String[500];    // Deklarataion
System.out.println( wert[i] );      // Inhalt des i-ten Elements ausgeben
```

Der Einstieg

Mac-BenutzerInnen können diesen Teil überspringen und den Java-Compiler einfach an der Kommandozeile (Terminal) aufrufen: Finder > 'Programme' > 'Dienstprogramme' > '**Terminal**'

Für Windows laden Sie die Datei **JavaKurs.ZIP** herunter und entpacken Sie die ZIP-Datei. Alle Dateien, die für den Kurs relevant sind, auch Ihre eigenen Java-Programme, kommen in diesen Ordner **JavaKurs**.



Öffnen Sie eine **Eingabeaufforderung**, und wechseln Sie in den Ordner **JavaKurs**. Falls Sie sich nicht mehr an die entsprechenden Kommandozeilen-Befehle erinnern (Laufwerkswechsel, `dir`, `cd`), lesen Sie bitte im Skriptum nach.

Beispiel:

```
C:\>  U:
U:\>  dir
U:\>  cd  EinfDV
U:\EinfDV>  dir
U:\EinfDV>  cd JavaKurs
U:\EinfDV\JavaKurs>
```

Wenn an der Kommandozeile ein Befehl eingegeben wird, sucht Windows zuerst im aktuellen Verzeichnis nach einem entsprechenden Programm, und falls hier keines existiert, in den in der Umgebungsvariablen PATH angegebenen Ordnern. Falls Windows dort auch nicht fündig wird, gibt es folgende Fehlermeldung beim Aufruf des Java-Compilers javac.exe:

```
U:\EinfDV\JavaKurs>  javac  HalloWelt.java
Der Befehl "javac" ist entweder falsch geschrieben oder
konnte nicht gefunden werden.
```

Die für die Java-Entwicklung wichtigen Programme stehen in Ordner `sdk/bin` oder `jdk/bin`. Sie können den Pfad händisch eingeben:

```
U:\EinfDV\JavaKurs>  sdk\bin\javac  HalloWelt.java
U:\EinfDV\JavaKurs>
```

Die (relative) Pfadangabe **sdk\bin\javac** (oder **.\sdk\bin\javac**) funktioniert. Da Sie sich die letzten Befehle auch immer mit der Pfeil ↑ - Taste holen können, spielt auch der Tippaufwand keine Rolle.

Eine bequemere Variante ist jedoch, die Pfadangabe zur PATH-Variable hinzuzufügen. Die Eingabe von `path` allein zeigt den derzeitigen Wert an:

```
U:\EinfDV\JavaKurs>  path
PATH=C:\Windows\system32;C:\Program Files;
```

Sie können nun einen neuen Wert für die PATH-Variable setzen, wobei man geschickterweise den bisherigen Inhalt wieder mit einbaut (%path%).

```
U:\EinfDV\JavaKurs>  path .;sdk\bin\;%path%
```

Um das weiter zu vereinfachen, ist im Ordner `JavaKurs` ein Batch-Progrämmchen namens **PATH_setzen.bat** vorbereitet das temporär im schwarzen Fenster die PATH-Variable anpasst. Daher nicht vergessen:

```
U:\EinfDV\JavaKurs> PATH_setzen
```

Am Beginn jeder Programmiersitzung in den Ordner JavaKurs wechseln und zuerst: PATH_setzen ↵ ausführen!

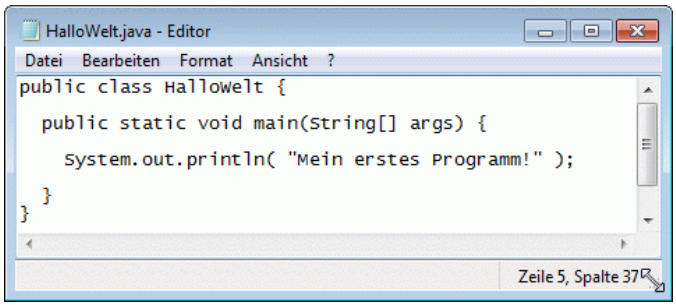
Hinweise für Mac-BenutzerInnen:

Der Pfad zum Java-Compiler und Interpreter ist automatisch gesetzt.

TextEdit ermöglicht das Bearbeiten einfacher Textdateien (entspricht dem Editor (Notepad) in Windows).

Finder > 'Programme' > 'TextEdit'

WICHTIG: 'Format' > **'In reinen Text umwandeln'** !!! und beim Speichern natürlich auf die Extension .java achten.




```
U:\EinfDV\JavaKurs> notepad  HalloWelt.java
```

Im Ordner `JavaKurs` gibt es schon ein Java-Quell-Programm (**source code**) namens `HalloWelt.java`. Öffnen Sie diese Datei mit dem ganz normalen Windows-Editor (notepad.exe), praktischerweise natürlich gleich per Kommandozeilenbefehl.

Mac-UserInnen können dafür TextEdit verwenden.

Nun kompilieren Sie diese Datei:
Der Java-**Compiler** heißt `javac`, und er braucht als Argument natürlich den Namen der Datei, die er compilieren soll (wichtig: Dateiname hier *mit* Extension!)

`javac HalloWelt.java` ↵

Soferne keine Fehler im Programm sind, redet der Compiler nicht viel, sondern konzentriert sich auf seine Arbeit: er erzeugt aus dem Quellcode maschinenunabhängigen **Bytecode**. Der Bytecode steht dann in der Datei `HalloWelt.class`.

```
U:\EinfDV\JavaKurs>dir *.class

U:\EinfDV\JavaKurs> javac HalloWelt.java

U:\EinfDV\JavaKurs>dir *.class

15.03.2010  10:07    438 HalloWelt.class
```

Um das Programm (den Bytecode) auszuführen, bemühen wir den Java-**Interpreter**.

`java HalloWelt` ↵

(der Java-Interpreter will das unbedingt *ohne* die Extension .class!)

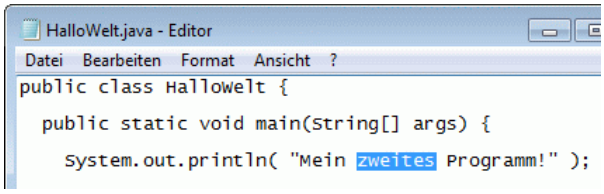
```
U:\EinfDV\JavaKurs> java HalloWelt
Mein erstes Java-Programm!
U:\EinfDV\JavaKurs>
```

So, nun können wir uns in den Entwicklungszyklus stürzen:

1. Quellcode im Editor verändern (z.B. "mein zweites … "),
   Speichern nicht vergessen (am schnellsten mit Strg+S Mac OS: cmd ⌘ + S)
2. Kompilieren: **`javac Xyz.java`**
   falls Fehler auftreten: Fehlermeldung studieren, Quellcode studieren, Unterlagen studieren, …, Lehrer fragen
3. Ausführen: **`java Xyz`**
   falls das Programm nicht tut, was Sie erwartet hatten: Unterlagen studieren, …, Lehrer fragen
4. weiter bei 1.

Beispiel (01): Ändern Sie den Quelltext (source code) des HalloWelt.java-Programms,
speichern Sie,
kompilieren Sie die Datei,
und führen Sie sie den Bytecode aus:

```
U:\EinfDV\JavaKurs>  notepad  HalloWelt.java
U:\EinfDV\JavaKurs>  javac  HalloWelt.java
U:\EinfDV\JavaKurs>  java  HalloWelt
Mein zweites Programm!
```



Beispiel (02): Was passiert, wenn man einen Strichpunkt (Semikolon) vergisst? (Das wird nämlich in Ihrer Programmierer-Karriere öfters passieren ;-)
Speichern, kompilieren, ausführen
Der Compiler erwartet in Zeile 4 ein ';', das fehlt nämlich am Ende von Zeile 3. Es handelt sich hier um einen **Syntax-Fehler**.

```
U:\EinfDV\JavaKurs>  type HalloWelt.java
public class HalloWelt {
    public static void main(String[] args) {
    System.out.println("Mein zweites Programm")
    }
}

U:\EinfDV\JavaKurs> javac HalloWelt.java
HalloWelt.java:4: ';' expected
    }
    ^
1 error
```

Programmierfehler

Beim Programmieren können dem/der Programmierer/in **logische Fehler** sowie **syntaktische Fehler** unterlaufen. Syntax-Fehler sind ein Verstoß gegen "Rechtschreibung und Grammatik" der Programmiersprache (entweder, weil der/die Programmmierer/in die Syntax nicht beherrscht, oder in den meisten Fällen einfach Tippfehler). Logische Fehler sind z.B. Denkfehler in der Ablauflogik, oder wenn der/die Programmierer/in nicht alle Möglichkeiten bedenkt. Ein Compiler findet syntaktische Fehler (wenn der Compiler nicht versteht, was Sie in Java sagen wollen, kann er es auch nicht in Maschinensprache übersetzen), aber keine Compiler kann logische Fehler erkennen; Wenn Sie Unsinn programmieren, macht das Programm genau den programmierten Unsinn.
Auf Englisch werden Programmfehler bugs (Ungeziefer) genannt, Fehlersuche nennt man daher debugging.

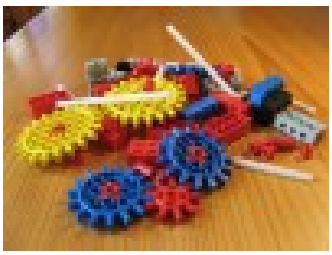
Fehlermeldungen des Compilers / Syntaxfehler:

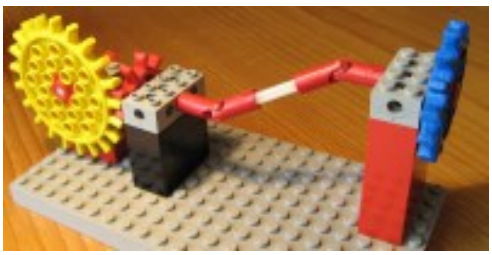
- Sehen Sie sich immer nur den ersten Fehler an, und versuchen Sie diesen zu beheben; die weiteren Fehlermeldungen ergeben sich oft aus dem ersten Fehler (Folgefehler), und verschwinden mit dessen Behebung.
- `';' expected` z.B. heißt: der Compiler hat einen Strichpunkt erwartet; höchstwahrscheinlich haben Sie den vergessen (einer der häufigsten Fehler);
- Sie bekommen immer eine Zeilennummer angezeigt, die Ihnen hilft, die Stelle mit dem Problem zu finden. Bei `';' expected` sollten Sie aber in der Zeile darüber suchen.
- Wenn der Compiler eine Datei nicht findet, hat er meistens recht: sehen Sie mit `dir` nach, und kontrollieren Sie, wohin und mit welcher Extension Sie Ihren Quellcode gespeichert haben.

Zum Thema Programmierfehler folgender Auszug aus: http://de.wikipedia.org/wiki/Programmfehler

Folgen von Programmfehlern

- 1962 führte ein fehlender Bindestrich in einem FORTRAN-Programm zum Verlust der Venus-Sonde Mariner 1, welche über 80 Millionen US-Dollar gekostet hatte.
- Zwischen 1985 und 1987 gab es mehrere Unfälle mit dem medizinischen Bestrahlungsgerät Therac-25. Infolge einer Überdosis, die durch fehlerhafte Programmierung und fehlende Sicherungsmaßnahmen verursacht wurde, mussten Organe entfernt werden, ein Patient verstarb drei Wochen nach der Bestrahlung.
- 1996 wurde der Prototyp der Ariane 5-Rakete der Europäischen Raumfahrtbehörde eine Minute nach dem Start zerstört, weil Programmcode, der von der Ariane 4 übernommen wurde und nur für einen (von der Ariane 4 nicht überschreitbaren) Bereich funktionierte, die Steuersysteme zum Erliegen brachte, nachdem ebendieser Bereich von der Ariane 5 überschritten wurde.
- 1999 verpasste die NASA-Sonde *Climate Orbiter* den Landeanflug auf den Mars, weil die Programmierer das falsche Maßsystem verwendeten – Yard statt Meter. Die NASA verlor dadurch die mehrere hundert Millionen Dollar teure Sonde.





Wenn Sie beginnen zu Programmieren, wechseln Sie die Seiten: vom Lager der Anwender (user) ins Lager der Entwickler (developer). **Als Entwickler/in müssen Sie beginnen, ingenieurmäßig zu denken.** Es gibt keine Schritt-für-Schritt-Anleitungen, sondern Sie sind diejenige/derjenige, die/der Lösungen findet und Verfahren entwickelt. Sie werden daher viele Aufgaben bekommen, wo Sie herausgefordert sind, relativ selbständig Lösungen zu entwickeln. Wenn Sie dabei Fehler machen, ist das sehr gut – machen Sie Fehler, und lernen Sie daraus! Versuchen Sie nicht, auf einmal komplizierte Strukturen oder komplizierte Ausdrücke zu schreiben, denn das ist sehr fehlerträchtig, sondern gehen Sie **Schritt für Schritt** vor, und **testen Sie jeden einzelnen Zwischenschritt!!!**

Sollten Sie andere Personen um Hilfe bei der Fehlersuche bitten (Helpdesks, Foren etc.), so beachten Sie bitte folgendes:

„Mein Programm läuft nicht" ist keine qualifizierte Meldung, damit kann niemand was anfangen, das ist so ähnlich wie: „mein Computer geht nicht".

- Bekommen Sie beim Kompilieren eine Fehlermeldung, so kann eine Bitte um Hilfe so ausschauen:
  „Beim Kompilieren bekomme ich folgende Fehlermeldung:" und dann folgt die VOLLSTÄNDIGE (!) Fehlermeldung und natürlich der vollständige Quelltext.
  In diesem Fall liegt wohl ein **Syntax-Fehler** vor (ungültiges Sprachkonstrukt, z.B. Klammer fehlt etc.).
- Bekommen Sie beim Ablauf des Programms einen Fehler (**Laufzeit-Fehler**, **runtime error**), liegt im Allgemeinen ein Logikfehler vor, der vielleicht nur bei bestimmten Eingaben auftritt. Daher ist es wichtig, nicht nur die vollständige Fehlermeldung und den vollständigen Quelltext anzugeben, sondern auch die (Eingabe-)Daten, die verarbeitet wurden.
- Sie bekommen keine Fehlermeldung, aber das Programm tut nicht das, was Sie erwarten. In diesem Fall ist die Aussage: „Mein Programm läuft nicht" nicht korrekt, denn es läuft ja. Es liegt höchstwahrscheinlich ein **Logikfehler**, ein Denkfehler des Programmierers / der Programmiererin vor. Geben Sie daher zusätzlich an, was Sie erwartet haben, und was das Programm stattdessen tatsächlich tut.

Übungen zu Variablen, Operatoren, Ausdrücken:

Beispiel (03): Verwenden Sie die Datei 'HalloWelt.java' als **Grundgerüst** für alle weiteren Beispiele. Jedes Mal, wenn Sie ein neues Beispiel beginnen, erstellen Sie einfach mittels Explorer oder copy eine Kopie von 'HalloWelt.java', und bearbeiten diese dann. Vergessen Sie nicht, dass der Klassenname immer mit dem Dateinamen übereinstimmen muss!

```
U:\EinfDV\JavaKurs>  copy HalloWelt.java  beispiel02.java
U:\EinfDV\JavaKurs>  notepad  beispiel02.java
```

Beispiel (04): Machen Sie sich mit Java vertraut, indem Sie ein bisschen experimentieren: Tippen Sie Java-Quellcode ein, lassen Sie das Programm laufen, versuchen Sie, den Sinn zu verstehen, ändern Sie den Quellcode ab, schauen Sie, was dann passiert, versuchen Sie wiederum zu verstehen, warum das passiert, was passiert …. .

```
public class wertzuweisungen {
  public static void main(String[] args) {

  // Variablendeklaration
  int i, j, k;

  // Wertzuweisungen an Variable
  i = 5;
  j = 3;
  k = i + j;

  // Ausgabe am Bildschirm
  System.out.print("Inhalt von i ist: ");
  System.out.println(i);
  System.out.print("Inhalt von j ist: ");
  System.out.println(j);
  System.out.print("Inhalt von k ist: ");
  System.out.println(k);
  System.out.println("---");
  }
}
```

```
Inhalt von i ist: 5
Inhalt von j ist: 3
Inhalt von k ist: 8
---
```

Achten Sie auf Groß- und Kleinschreibung, denn Java ist **case sensitive**.
Weiters ist Java eine **streng typisierte Sprache**, bevor Sie einer Variablen Werte zuweisen können, müssen Sie die Variable **deklarieren**. Beachten Sie, dass der Klassenname mit dem Dateinamen übereinstimmen muss.

Anweisungen werden sequentiell von oben nach unten nacheinander abgearbeitet. Erweitern Sie obiges Beispiel wie folgt. Gehen Sie das Programm zuerst nur auf dem Papier durch, und tragen Sie nach jeder Anweisung in der Tabelle ein, wenn sich die Inhalte von Variablen ändern. Führen Sie das Programm aus, und kontrollieren Sie, ob auch tatsächlich das herauskommt, was Sie erwartet haben:

```
  i = 5;
  j = 3;
  k = i + j;
  System.out.print("Inhalt von i ist: ");
  System.out.println(i);
  System.out.print("Inhalt von j ist: ");
  System.out.println(j);
  System.out.print("Inhalt von k ist: ");
  System.out.println(k);
  j = 7;
  System.out.print("Inhalt von j ist: ");
  System.out.println(j);
  System.out.print("Inhalt von k ist: ");
  System.out.println(k);
  k = i + j;
  System.out.print("Inhalt von k ist: ");
  System.out.println(k);
  k = k + 1;
  System.out.print("Inhalt von k ist: ");
  System.out.println(k);
```

Wie ändern sich die Inhalte der Variablen während des Programmlaufs? Füllen Sie bitte die Tabelle *vor* dem Programmablauf *händisch* aus:

Anweisung	i	j	k
i = 5	5		
j = 3		3	
k = i + j			8
j = 7			
k = i + j			
k = k + 1			

Beispiel (05): Nun probieren Sie **Fließkommazahlen** aus. Achtung: **Dezimaltrennzeichen** ist immer der **Punkt**, nicht das Komma! Das Ergebnis der dritten Zeile ist durchaus überraschend, wenn man andere Programmiersprachen gewohnt ist:

```
  System.out.println( 1.0f / 3.0f );     // Division mit float-Genauigkeit
  System.out.println( 1.0 / 3.0 );       // Division mit double-Genauigkeit
  System.out.println( 1 / 3 );           // Division mit int-Genauigkeit
```

Beispiel (06): Berechnen Sie das arithmetische Mittel aus den Werten 3, 2, 5. Wenn das Ergebnis nicht Ihren Erwartungen entspricht, machen Sie sich bitte Gedanke über die **Datentypen**, die in dem Ausdruck vorkommen!

```
System.out.println( (3+2+5)/3 );
```

Beispiel (07): Experimentieren Sie ein bisschen mit Variablen, Operatoren, Ausdrücken, machen Sie sich damit vertraut:

```
System.out.println( 17 > 32 );          // Ergebnis vom Typ boolean
System.out.println( 1.0 / 3 );          // Ergebnis ist von welchem Typ?
System.out.println( 1 + 10 * 3 );       // Punktrechnung vor Strichrechnung
System.out.println( "ABC" + "DEF" );    // Zusammenhängen von Strings
System.out.println( "ABC" + 'Z' );      // Zusammenhängen von String u. Char
```

Beispiel (08): Und ein paar Übungen zum Umgang mit **Strings** (**Zeichenketten**). Der Operator '+' hängt Zeichenketten zusammen. Das Leerzeichen ist nicht nichts, sondern ein Zeichen wie jedes andere.


```
Inhalt der Variablen vorname ist: Max
Inhalt der Variablen nachname ist: Mustermann

Der volle Name ist: MaxMustermann

(ups, da gehoert ein Leerzeichen dazwischen)

Der volle Name ist: Max Mustermann
```

```
// Variablendeklaration;
// bei der Deklaration kann man gleich Werte zuweisen
String vorname = "Max";
String nachname = "Mustermann";
String vollername;

System.out.print("Inhalt der Variablen vorname ist: ");
System.out.println(vorname);
System.out.print("Inhalt der Variablen nachname ist: ");
System.out.println(nachname);
System.out.println();              // Ausgabe einer Leerzeile

// Der Operator + hängt Strings zusammen
vollername = vorname + nachname;
System.out.println("Der volle Name ist: " + vollername );
System.out.println();
System.out.println( "(ups, da gehoert ein Leerzeichen dazwischen)" );
System.out.println( );
vollername = vorname + " " + nachname;
System.out.println("Der volle Name ist: " + vollername );
```

Beispiel (09): Eingaben über die Tastatur sind in Java relativ umständlich zu realisieren, deshalb gibt es in diesem Kurs eine vorbereitete Klasse zum Einlesen von der Tastatur:

```
Gib bitte Deinen Namen ein: Max
Hallo Max, wie geht's?
```

```
String antwort;
System.out.println( "Gib bitte Deinen Namen ein: " );
antwort = Einlesen.String();
System.out.println( "Hallo " + antwort + ", wie geht's?");
```

`Einlesen.String()` liest einen String über die Tastatur ein. Wie Sie am deutschen Namen dieser Klasse erkennen können, handelt es sich *nicht* um einen Bestandteil von Java, es muss sich die Datei `Einlesen.class` im aktuellen Verzeichnis befinden. Details zum Einlesen von der Tastatur finden Sie weiter hinten auf Seite 27.

Beispiel (10): Schreiben Sie ein Programm, das Euro in Schilling umrechnet. Für das Einlesen von Zahlenwerten von der Tastatu verwenden Sie `Einlesen.Int()` oder `Einlesen.Double()`. Das Programm soll eine ausführliche Ausgabe machen (ausführlich heißt hier: die Benutzer sollen informiert werden, wie viel von welcher Währung umgerechnet in welche andere Währung wie viel ergibt, das Programm soll nicht nur eine Zahl hinschreiben, mit der die Benutzer nichts anfangen können!).

```
Bitte Euro-Betrag eingeben: 100
Der Umrechnungsfaktor ist: 13.76
100.0 Euro sind umgerechnet 1376.0 Schilling
```

```
double euro, umrechnungsfaktor=13.76, schilling;
System.out.print( "Bitte Euro-Betrag eingeben: " );
euro = Einlesen.Double();
System.out.println("Der Umrechnungsfaktor ist: " + umrechnungsfaktor );
schilling = euro * umrechnungsfaktor;
System.out.println(euro+" Euro sind umgerechnet "+schilling+"Schilling");
```

Beispiel (11): Gegeben sei ein Rechteck mit Länge und Breite. Schreiben Sie ein Programm, das Fläche und Umfang dieses Rechtecks ausgibt. Welche Variablen brauchen Sie? Gestalten Sie die Ausgabe ausführlich.

```
Bitte Laenge des Rechtecks eingeben: 25
Bitte Breite des Rechtecks eingeben: 57
Gegeben: Rechteck mit Laenge: 25 und Breite: 57
Umfang: 164.0
Flaeche: 1425.0
```

Beispiel (12): Die Klasse `Math` bietet eine Funktion zum Potenzieren x^y (engl. power): `Math.pow(x,y)`. Der Rückgabewert ist vom Typ double. (`Math` ist eine vordefinierte Klasse von Java, Ihr eigenes Programm darf daher *keinesfalls* Math.java heißen, Sie würden damit die Java-Klasse überlagern.)

```
System.out.println( Math.pow( 10, 6 ) );
```

```
double x=5; double y=2;
System.out.println( Math.pow(x,y) );
```

Beispiel (13): Die Klasse `Math` bietet eine vordefinierte Konstante für die Kreiszahl π. Der Rückgabewert ist vom Typ double.

```
System.out.print( Math.PI );
```

Beispiel (14): Schreiben Sie ein Programm, das für einen gegebenen Radius r den Kreisumfang und die Kreisfläche berechnet (U=2rπ, F=r^2π).
Für pi verwenden Sie ein Literal (3.14159) oder besser `Math.PI`

r^2 können Sie als r*r berechnen oder mit `Math.pow(r, 2)`

```
Radius: 10.0
Kreisumfang: 62.83185307179586
Kreisflaeche: 314.1592653589793
```

Beispiel (15): Im vorigen Beispiel haben Sie für das Potenzieren die Methode `pow` (power) der Klasse `Math` verwendet. Die Umkehrung bietet die Methode `Math.sqrt()` – square root, Quadratwurzel. Das Argument kommt in die runden Klammern. Probieren Sie es aus, und berechnen Sie die Quadratwurzel von 25.

$c = \sqrt{a^2 + b^2}$ sähe dann so aus: `c = Math.sqrt( Math.pow(a,2) + Math.pow(b,2) )`
Die Klasse `Math` bietet eine Vielzahl mathematischer Funktionen. Beispiele finden Sie auf der Seite 8 (Auszug aus der Original-Dokumentation).

Java-Dokumentation zur Klasse Math, zum Schmökern und Nachschauen, *nicht* zum Auswendiglernen.

java.lang http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Math.html

Class Math

Field Summary	Rückgabe-Datentyp / Funktionsname
static double	**E** The double value that is closer than any other to *e*, the base of the natural logarithms.
static double	**PI** The double value that is closer than any other to *pi*, the ratio of the circumference of a circle to its diameter.

Datentyp des Rückgabewertes

Funktionsname, in Klammer die Argumente (mit mögl. Datentyp)

Method Summary	
static double	**abs**(double a) Returns the absolute value of a double value.
static float	**abs**(float a) Returns the absolute value of a float value.
static int	**abs**(int a) Returns the absolute value of an int value.
static long	**abs**(long a) Returns the absolute value of a long value.
static double	**acos**(double a) Returns the arc cosine of an angle, in the range of 0.0 through *pi*.
static double	**asin**(double a) Returns the arc sine of an angle, in the range of -*pi*/2 through *pi*/2.
static double	**atan**(double a) Returns the arc tangent of an angle, in the range of -*pi*/2 through *pi*/2.
static double	**atan2**(double y, double x) Converts rectangular coordinates (x, y) to polar (r, *theta*).
static double	**ceil**(double a) Returns the smallest (closest to negative infinity) double value that is greater than or equal to the argument and is equal to a mathematical integer.
static double	**cos**(double a) Returns the trigonometric cosine of an angle.
static double	**cosh**(double x) Returns the hyperbolic cosine of a double value.
static double	**exp**(double a) Returns Euler's number *e* raised to the power of a double value.
static double	**floor**(double a) Returns the largest (closest to positive infinity) double value that is less than or equal to the argument and is equal to a mathematical integer.
static double	**log**(double a) Returns the natural logarithm (base *e*) of a double value.
static double	**log10**(double a) Returns the base 10 logarithm of a double value.
static double	**log1p**(double x) Returns the natural logarithm of the sum of the argument and 1.
static double	**max**(double a, double b) Returns the greater of two double values.
static float	**max**(float a, float b) Returns the greater of two float values.
static int	**max**(int a, int b) Returns the greater of two int values.
static long	**max**(long a, long b) Returns the greater of two long values.
static double	**min**(double a, double b) Returns the smaller of two double values.
static float	**min**(float a, float b) Returns the smaller of two float values.
static int	**min**(int a, int b) Returns the smaller of two int values.
static long	**min**(long a, long b) Returns the smaller of two long values.
static double	**pow**(double a, double b) Returns the value of the first argument raised to the power of the second argument.
static double	**random**() Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
static double	**rint**(double a) Returns the double value that is closest in value to the argument and is equal to a mathematical integer.
static long	**round**(double a) Returns the closest long to the argument.
static int	**round**(float a) Returns the closest int to the argument.

static double	**signum**(double d) Returns the signum function of the argument; zero if the argument is zero, 1.0 if the argument is greater than zero, -1.0 if the argument is less than zero.
static float	**signum**(float f) Returns the signum function of the argument; zero if the argument is zero, 1.0f if the argument is greater than zero, -1.0f if the argument is less than zero.
static double	**sin**(double a) Returns the trigonometric sine of an angle.
static double	**sinh**(double x) Returns the hyperbolic sine of a double value.
static double	**sqrt**(double a) Returns the correctly rounded positive square root of a double value.
static double	**tan**(double a) Returns the trigonometric tangent of an angle.
static double	**tanh**(double x) Returns the hyperbolic tangent of a double value.
static double	**toDegrees**(double angrad) Converts an angle measured in radians to an approximately equivalent angle measured in degrees.
static double	**toRadians**(double angdeg) Converts an angle measured in degrees to an approximately equivalent angle measured in radians.

Beispiel (16): **Explizite Typumwandlung** (**type cast**):
Falls Sie versuchen, einen Fließkommawert einer Ganzzahlvariablen zuzuweisen, würden eventuelle Nachkommastellen abgeschnitten (*possible* loss of precision), daher weigert sich der Compiler, weil er meint, dass Sie im Begriffe sind, einen Fehler zu begehen (die Rakete könnte wegen so was abstürzen). …):

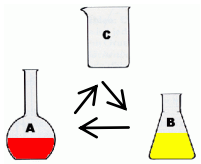
```
int i = 3.0/2.0;
```

```
Experimente.java:12: possible loss of precision
found   : double
required: int
        int i = 3.0/2.0;
                ^
1 error
```

Falls Sie wirklich die Nachkommastellen abschneiden wollen, müssen Sie eine explizite Typumwandlung (type cast) durchführen: Stellen Sie ein `(int)` vor den Ausdruck:

```
int i = (int) ( 3.0/2.0 );
```

Beispiel (17): Es gibt eine Variable a und eine Variable b. Die Inhalte der Variablen a und b sollen vertauscht werden. Sobald Sie aber mittels a=b den Wert von a überschreiben, ist dieser für immer verloren. Sie müssen also den originalen Wert von a daher irgendwo zwischenspeichern. Dazu benötigen Sie eine Hilfsvariable `hilfsvar`, um den Dreieckstausch durchzuführen. Schreiben Sie sich zur Verdeutlichung in die Tabelle, welchen Wert die Variablen zum jeweiligen Zeitpunkt haben.



Wie ändern sich die Inhalte der Variablen während des Programmlaufs? Füllen Sie bitte die Tabelle *vor* dem Programmablauf *händisch* aus:

Anweisung	a	b	hilfsvar
a = 1	1		
b = 3		3	

Beispiel (18): `Math.random()` erzeugt eine **Zufallszahl** vom Typ double im Bereit größer gleich 0 und kleiner 1. Führen Sie das Programm einige Male aus.

```
System.out.println( Math.random() );
```

Beispiel (19): Programmieren Sie einen **Würfel**, der die Zahlen 1 bis 6 liefert.
`Math.random()` liefert Zahlen im Bereich von 0.0000 bis 0.9999 (immer kleiner als 1.)
1.) Führen Sie das Programm mehrmals hintereinander aus!
2.) Wenn Sie mit 6 multiplizieren, erhalten Sie Werte im Bereich von 0.0000 bis 5.9999 . Führen Sie das Programm mehrmals hintereinander aus.
3.) Um die Nachkommastellen abzuschneiden, führen Sie eine explizite Typkonversion (type cast, siehe Beispiel (16): Seite 10) in einen Ganzzahltyp durch: stellen Sie ein `(int)` vor den gesamten Ausdruck, dessen Ergebnis in eine Ganzzahl umgewandelt werden soll. Probieren Sie es aus! Nun erhalten Sie Werte im Bereich von 0 bis 5.
4.) Jetzt brauchen Sie nur 1 addieren, und erhalten Werte im Bereich von 1 bis 6.

Beispiel (20): Inkrement-Operator: Der Operator ++ erhöht den Inhalt der variablen um 1, `i++` hat also die gleiche Wirkung wie `i=i+1`. Probieren Sie es aus.

```
i=5;
System.out.println( i );
i=i+1;
System.out.println( i );
i++;
System.out.println( i );
```

Zusatzbeispiele für Fortgeschrittene

Beispiel (21): Runden auf 3 Dezimalstellen:

```
System.out.println( Math.round( Math.PI * 1000) / 1000.0);
```

Beispiel (22): Zahlensystemumwandlung, Verschachtelung von Funktionen:
Geben Sie die Dezimalzahl 42 als Binärzahl dargestellt aus:

```
System.out.println( Integer.toBinaryString( 42 ) );
```

Beispiel (23): `i+=5` ist eine Kurzform von `i=i+5`. Das gibt es auch mit -=, *=, /=. Ob solche Formulierungen Programme besser lesbar machen, ist allerdings fraglich.

Beispiel (24): Modulo-Operator (ganzzahliger Divisionsrest): %

Beispiel für Modulo 5

7%5 ergibt 2 (7 durch 5 ergibt 1 und Divisionsrest 2)

```
 0  1  2  3  4  5  6  7  8  9 10 11 12 13 …
 ↓              ↓              ↓
 0  1  2  3  4  0  1  2  3  4  0  1  2  3 …
```

Damit lassen sich mit Strings oder Arrays ringförmige Datenstrukturen realisieren, dazu siehe später.

Weiters können Sie damit feststellen, ob der Inhalt der Variablen int z eine gerade Zahl ist: wenn z%2==0 dann ist z durch 2 teilbar.

Strings (Zeichenketten)

String-Literale (in doppelten Hochkommas) und String-Verkettung (mit dem Operator +) haben Sie schon kennengelernt. Auf den nächsten Seiten finden Sie Möglichkeiten beschrieben, wie Strings bearbeitet werden können.

(`String` ist eine vordefinierte Klasse von Java, Ihr eigenes Programm darf daher *keinesfalls* String.java heißen, Sie würden damit die Java-Klasse überlagern. Da String in Java eine Klasse und kein primitiver Datentyp ist, wird **S**tring mit großen 'S' geschrieben.)

Beispiel (25): Länge eines Strings feststellen: `length()` ist eine Methode eines String-Objekts, das die Länge eines Strings zurückgibt.

```
String s = "Max Muster";
System.out.print( "Der String '" );
System.out.print( s );
System.out.print( "' ist " );
System.out.print( s.length() );
System.out.print( " Zeichen lang." );
System.out.println();
```

```
Der String 'Max Muster' ist 10 Zeichen lang
```

Die Syntax `s.length()` (als Methode des Strings s) ist etwas gewöhnungsbedürftig, aus anderen Programmiersprachen wäre man vielleicht `length(s)` gewohnt.
Strings können auch die Länge Null haben: Leerstring: `String s = "";`

Beispiel (26): `toLowerCase()` wandelt in Kleinbuchstaben um, `toUpperCase()` wandelt in Großbuchstaben um:

```
String s1 = "Max Muster";
System.out.println( s1.toLowerCase() );
System.out.println( s1.toUpperCase() );
```

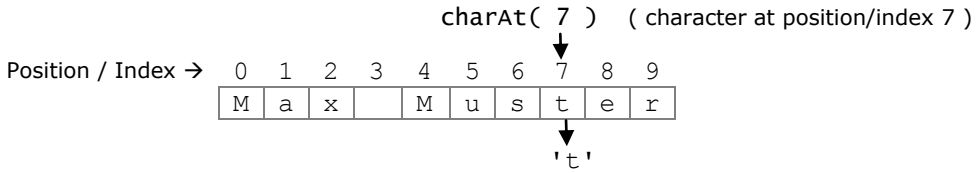
```
max muster
MAX MUSTER
```

Beispiel (27): `replace()` ersetzt Zeichen

```
String nonsens1, nonsens2;
nonsens1 = "blablabla";
nonsens2 = nonsens1.replace('a', 'u');
System.out.println(nonsens1);
System.out.println(nonsens2);
```

```
blablabla
blublublu
```

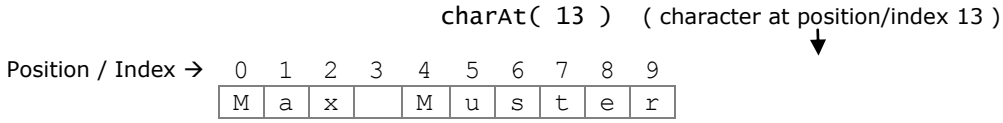
Beispiel (28): Welches Zeichen steht an Position 7 (hat den Index 7) ? `charAt()` liefert die Antwort.
Wichtig: Bei der Position von Zeichen (Index) wird immer bei Null zu zählen begonnen!
Der String "Max Muster" hat die Länge 10, aber die Indizes laufen von 0 bis 9 (nicht bis 10)!




```
//                  0123456789    <- Indices
String meinstring = "Max Muster";
int i=7;
System.out.print( "Das Zeichen an Position " );
System.out.print( i );
System.out.print( " ist ein '" );
System.out.print( meinstring.charAt(i) );
System.out.print( "'" );
System.out.println();
```

```
Das Zeichen an Position 7 ist ein 't'
```

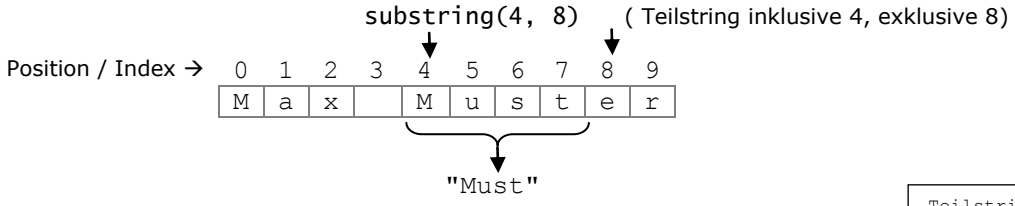
Beispiel (29): Welches Zeichen steht an Position 13 (hat den Index 13) ? `charAt( 13)` führt zu einem Laufzeitfehler: der String ist nur 10 Zeichen lang ist, daher ist der größte zulässige Index 9. Ein Index von 13 führt liefert eine Fehlermeldung: 'String index out of range' .




```
System.out.println( s.charAt( 13)  );
```

```
StringIndexOutOfBoundsException: String index out of range: 13
```

Beispiel (30): Nun soll ein Teilstring "herausgeschnitten" werden, z.B. ab Position 4 (bis *exklusive* Position 8). Eine Aufgabe für die Methode `substring`:

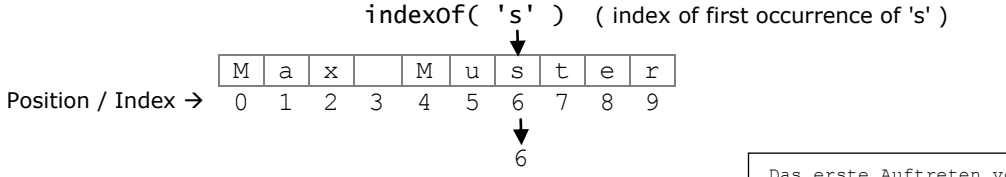



```
Teilstring: Must
```

```
meinstring = "Max Muster";
System.out.println("Teilstring: "+meinstring.substring(4, 8) );
```

Wenn nur 1 Argument angegeben wird, wird ab angegebenem Index bis zum Ende des Strings zurückgegeben.
`substring(4)` (Teilstring inklusive 4 bis zum Ende des Strings)

Beispiel (31): An welcher Position tritt das erste Mal das Zeichen 's' auf? Anders gefragt: welchen Index hat das erste Vorkommen von 's'? `indexOf( 's' )` liefert die Antwort ('O' in indexOf ist großgeschrieben).
Falls das gesuchte Zeichen gar nicht vorhanden ist, wird -1 zurückgegeben.




```
Das erste Auftreten von 's' ist an
Position 6.
```

```
meinstring = "Max Muster";
String suche_nach="s";
int p = meinstring.indexOf(suche_nach);
System.out.print("Das erste Auftreten von '" + suche_nach + "' ");
System.out.println("ist an Position " + p );
```

Übungen

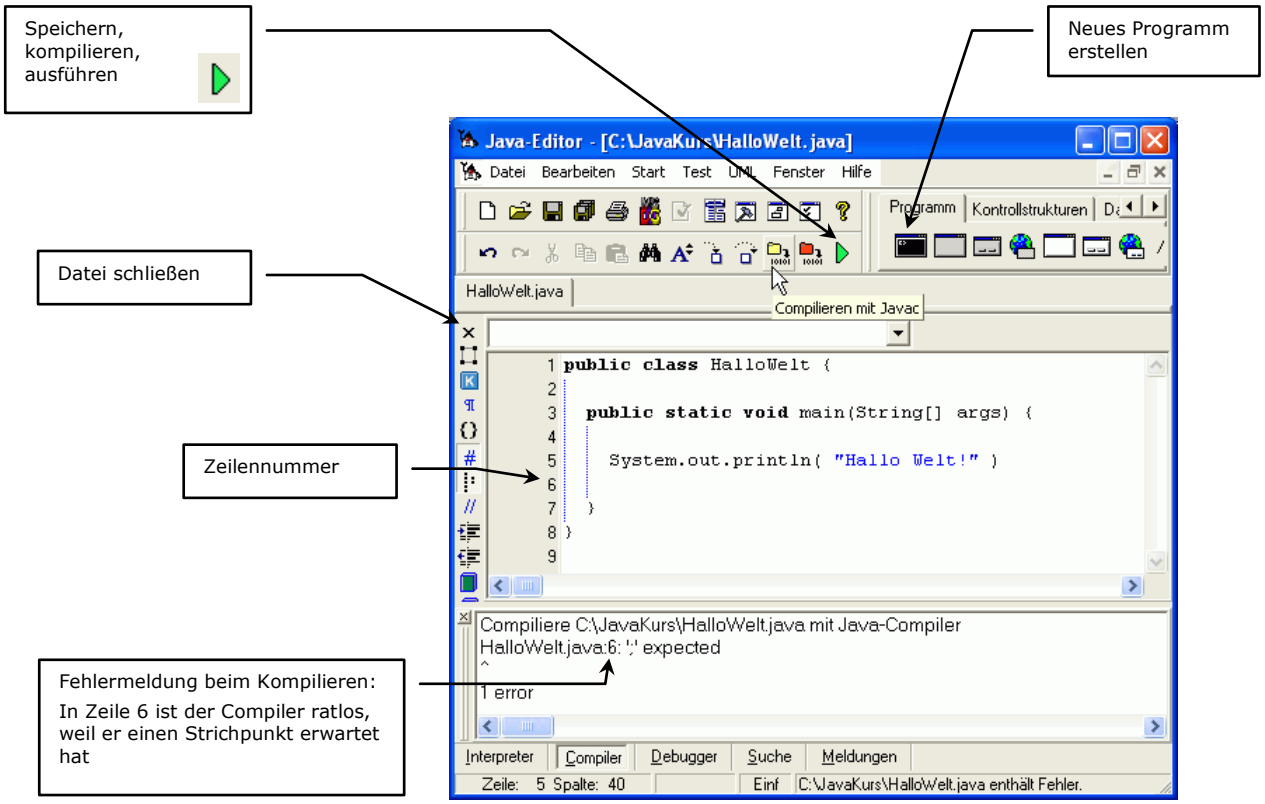
Beispiel (32): In einem String befindet sich ein Name, z.B. `String name = "Max Mustermann";`
Führen Sie nun zwei weitere Variablen `vorname` und `nachname` ein. Der String `name` soll jetzt mit Hilfe der Methode `substring`, die auf den String `name` angewendet wird, zerlegt werden: Der Vorname soll in der Variablen `vorname` gespeichert werden, der Nachname in der Variablen `nachname`. (Da Sie feste Indices eingeben, funktioniert das natürlich nur für diesen Namen. Die allgemeinere Lösung werden Sie im nächsten Beispiel angehen.)

Beispiel (33): Erweitern Sie die Funktionalität der vorherigen Beispiels. Das Prgrämmchen soll beliebige Namen in Vor- und Nachname zerlegen können: Als Trennzeichen fungiert natürlich das Leerzeichen. Also soll das Programm die Position des ersten Leerzeichens im Strings bestimmen. Mit dieser Position des Leerzeichens lassen sich die entsprechenden Teilstrings ermitteln.

Die String-Funktionen sind äußerst praktisch, werden auch in sehr vielen der folgenden Beispielen verwendet.

Java-Editor

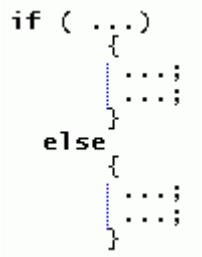
Es ist für das Verständnis wichtig, den Zyklus vom Schreiben von Quelltext (`.java`) mit einem Editor, dem Compilieren in maschinenunabhängigen Bytecode (`.class`), und dem Ausführen mittels Java-Interpreter praktiziert zu haben. Ab nun machen wir uns aber das Leben ein bisschen einfacher, und verwenden eine handliche kleine Entwicklungsumgebung für Lehrzwecke namens 'Java-Editor' (http://javaeditor.org/), die Sie im Ordner `JavaKurs` finden und mittels Doppelklick aufrufen können.
(Für größere Projekte gibt es Entwicklungsumgebungen (IDE, integrated development environment) wie z.B. **Eclipse** (http://de.wikipedia.org/wiki/Eclipse_(IDE)) oder **NetBeans** (http://de.wikipedia.org/wiki/NetBeans_IDE), die aber für unsere ersten Schritte etwas übertrieben wären.)



Praktische **Tastenkombinationen**:

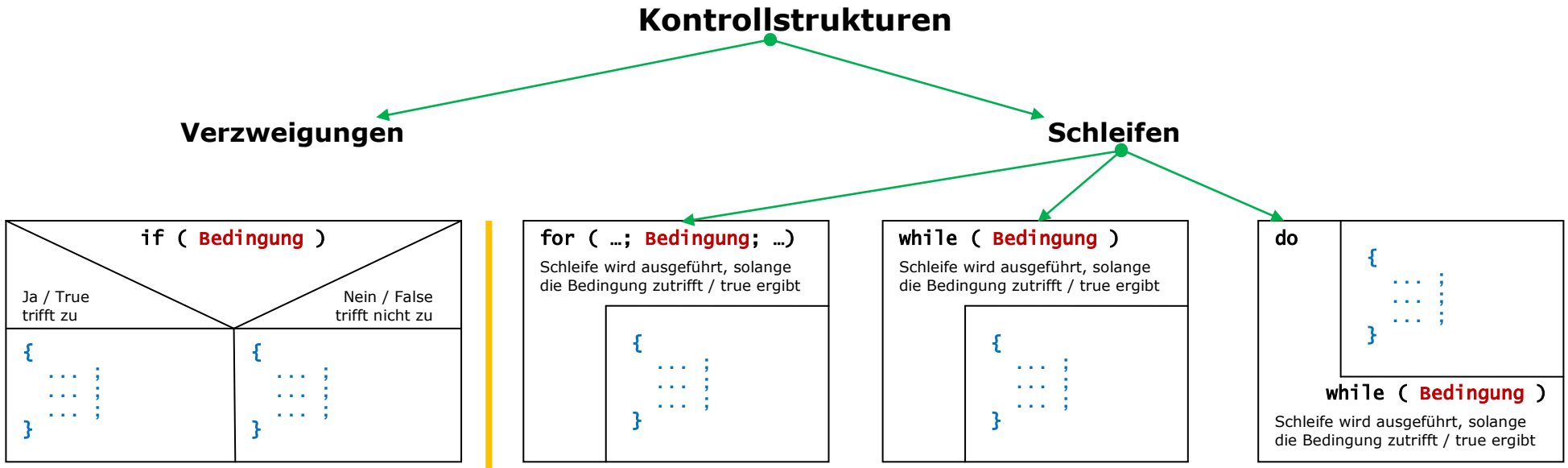
Strg+U	fügt an der Cursorposition `System.out.println()` ein
Strg+S	speichern
Strg+F9	kompilieren
F9	speichern, kompilieren und ausführen

Der Editor markiert zusammengehörige Klammerpaare und zeichnet gepunktete blaue Linien zwischen zusammengehörigen geschwungenen Klammern. Das soll Ihnen helfen, Blöcke korrekt einzurücken. Nehmen Sie diese Hilfe an!



Kontrollstrukturen, Steueranweisungen

Normalerweise werden Anweisungen einfach hintereinander (sequentiell) ausgeführt. Nun gibt es **Steueranweisungen / Kontrollstrukturen**, die diese sequentielle Ausführung abändern: **Verzweigung** und **Schleifen**.
Darstellung in Form von **Struktogrammen** (Nassi-Shneiderman-Diagrammen).




```
if ( Bedingung )
        { Anweisungsblock }
    else
        { Anweisungsblock }
```

Der else-Zweig kann auch entfallen.

```
for (Initialisierung der Lauf-
     variablen;
     Bedingung;
     Inkrement/Dekrement
     der Laufvariablen )
     { Anweisungsblock, der
     wiederholt wird }
```

Typischer Anwendungsfall für for-Schleifen sind Zählschleifen (Anzahl der Durchläufe bekannt)

```
while ( Bedingung )
      { Anweisungsblock, der
      wiederholt wird }
```

Allgemeinste Form der Schleife

Die Bedingung wird geprüft, *bevor* der Anweisungsblock wiederholt wird. Das heißt, dass Schleifen dieses Typs auch Null mal durchlaufen werden können.

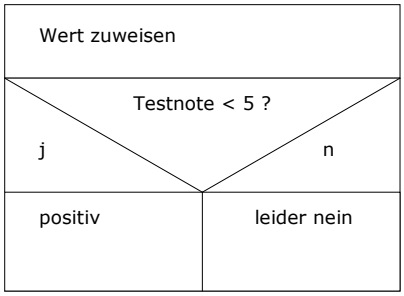
```
do { Anweisungsblock, der
     wiederholt wird }
   while ( Bedingung )
```

Die Bedingung wird erst *nach* dem Durchlaufen des Anweisungsblocks geprüft. Das heißt, dass Schleifen dieses Typs mindestens einmal durchlaufen werden.
Typischer Anwendungsfall ist das Warten auf eine bestimmte Benutzereingabe oder z.B. ein (Würfel-)Ergebnis.

Verzweigung, Alternative, bedingte Ausführung, Fallunterscheidung

```
public class Testnoten {

  public static void main(String[] args) {

    int testnote;
    System.out.print("Bitte Testnote eingeben: ");
    testnote = Einlesen.Int();

    if ( testnote < 5 )
        {
         System.out.println("positiv");
        }
      else
        {
         System.out.println("leider nein");
        }
  }
}
```

Struktogramm:



Wenn die Bedingung in der runden Klammer zutrifft (`true` ergibt), wird der anschließende, in geschwungener Klammer stehende Block ausgeführt. Trifft die Bedingung nicht zu (Ausdruck in runder Klammer ergibt `false`), wird der auf `else` folgende Block ausgeführt (der `else`-Zweig kann auch entfallen).

```
if ( ...)
      {
       ...;
       ...;
      }
  else
      {
       ...;
       ...;
      }
```

Einrücken trägt sehr wesentlich zur Übersichtlichkeit und Verständlichkeit bei!!

Der Editor zeichnet gepunktete blaue Linien zwischen zusammengehörigen geschwungenen Klammern. Das soll Ihnen helfen, Blöcke korrekt einzurücken. Nehmen Sie diese Hilfe an!

Es empfiehlt sich auch, beim Programmieren zuerst ein leeres, aber vollständiges Statement hinzuschreiben, dann kann man keine Klammer vergessen. Anschließend befüllt man die runde Klammer mit der Bedingung, und die Blöcke in den geschwungenen Klammern mit entsprechenden Anweisungen.

Mac OS X: geschwungene Klammern: { mit ⌥ + 8, } mit ⌥ + 9

Beispiel (34): Schreiben Sie ein Prgrämmchen, das den Benutzer nach der Testnote fragt, und danach ausgibt, ob die Note positiv oder negativ ist.

Beispiel (35): Schreiben Sie ein Prgrämmchen, das einmal würfelt (siehe Beispiel (19): Seite 11), die gewürfelte Zahl ausgibt, und dann dem Benutzer gratuliert, falls ein Sechser gewürfelt wurde, oder anderenfalls den Benutzer auffordert, noch einmal zu würfeln.

Vergleichsoperatoren:

==	gleich (!!Achtung, Fehlerquelle: **2** Gleichheitszeichen! 1 Gleichheitszeichen ist eine Wertzuweisung!!!)	
!=	ungleich	(Hinweis: in vielen Programmiersprachen wird <> für ungleich verwendet)
<	kleiner	
<=	kleiner oder gleich	
>	größer	
>=	größer oder gleich	

Da Strings in Java keine primitiven Datentypen sind (String großgeschrieben → Klasse), muss man zum Vergleichen von Strings die dafür vorgesehenen Methoden verwenden: `equals`, `equalsIgnoreCase`
Bei Vergleich mit `equals` müssen die Strings auch in Groß- und Kleinschreibung übereinstimmen, `equalsIgnoreCase` ignoriert Groß-u.Kleinschreibung.

```
String s="ENDE";
System.out.println( s.equals("ende") );                 liefert false
System.out.println( s.equalsIgnoreCase("ende") );       liefert true
```

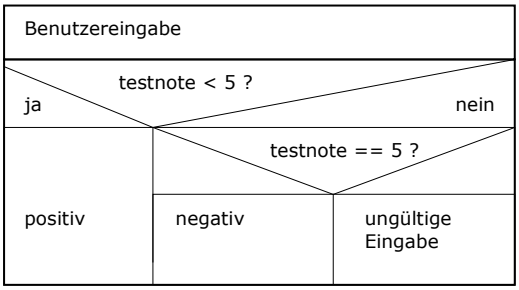
Beispiel (36): Der Benutzer soll eine Länge und eine Breite eingeben. Falls also der Inhalt der Variable `laenge` kleiner ist als der Inhalt von `breite`, sollen die Inhalte von `laenge` und `breite` vertauscht werden. Dazu benötigen wir eine Hilfsvariable `hilfsvar`, um den Dreieckstausch durchzuführen. Schreiben Sie sich zur Verdeutlichung in die Tabelle, welchen Wert die Variablen zum jeweiligen Zeitpunkt haben.

laenge	breite	hilfsvar

Beispiel (37): `if`s können auch verschachtelt werden, das heißt, in einem der Anweisungsblöcke in geschwungenen Klammer können natürlich wieder ifs stehen.

Dem Compiler sind Einrückungen egal, aber als Mensch sollten Sie **auf jeden Fall deutlich und sinnvoll einrücken**, um hier die Übersicht nicht zu verlieren. Der Java-Editor hilft Ihnen hier auch, indem blaue Linien zwischen die geschwungenen Klammerngemalt werden. Nehmen Sie diese Hilfe an!

```
if ( testnote < 5 )
        {
        System.out.println("positiv");
        }
    else
        {
        if ( testnote == 5 )
                {
                System.out.println("leider negativ");
                }
            else
                {
                System.out.println("ungueltige Eingabe");
                }
        }
```



Beispiel (38): In der Variablen `groesse` steht die Körpergröße. In der Variablen `mindest` steht die Mindestkörpergröße. Das Programm soll nun prüfen, ob die Mindestkörpergröße überhaupt erreicht ist oder nicht. Falls die Mindestkörpergröße nicht erreicht ist, soll das Programm nur ausgeben: "dafür sind Sie noch zu klein". Anderenfalls soll das Programm prüfen, ob die Mindestkörpergröße um mehr als 5cm überschritten wurde, und in diesem Fall ausgeben: "passt", anderenfalls "das war knapp".

a) Zeichnen Sie *zuerst* ein **Struktogramm**
b) codieren Sie anschließend die Lösung in Java

Sie können logische Ausdrücke auch mit UND: && und ODER: || verknüpfen bzw. mit NICHT: ! verneinen.

Ein bisschen Theorie (**Boole'sche Algebra**):
Logisch WAHR (TRUE, 1): Bedingung trifft zu
Logisch FALSCH (FALSE, 0): Bedingung triff nicht zu
Logische ODER-Verknüpfung: WAHR (TRUE, 1), wenn entweder die eine Bedingung zutrifft, oder die andere, oder beide. Nur dann FALSCH (FALSE, 0), wenn beide Bedingungen FALSCH.
Logische UND-Verknüpfung: Nur dann WAHR (TRUE, 1), wenn sowohl die eine Bedingung zutrifft, als auch die andere. Immer FALSCH, wenn auch nur eine Bedingung FALSCH.

Logisches ODER (OR)

0 ODER 0 → 0
0 ODER 1 → 1
1 ODER 0 → 1
1 ODER 1 → 1

Logisches UND (AND)

0 UND 0 → 0
0 UND 1 → 0
1 UND 0 → 0
1 UND 1 → 1

Negation (NOT)

NICHT 0 → 1
NICHT 1 → 0

Beispiel: Bedingungen ODER-verknüpft (OR, in Java: ||)

```
if ( note < 0  ||  note > 5 )   // wenn note größer 5 ODER note kleiner 1
        {
        System.out.println("keine gültige Note, ist wohl ein Tippfehler?" );
        }
```

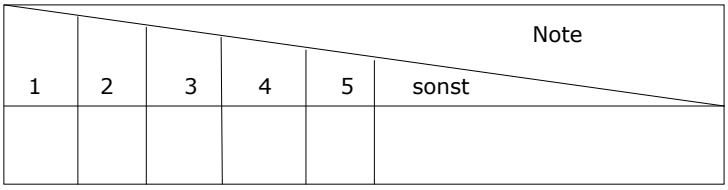
Beispiel: Bedingungen UND-verknüpft (AND, in Java: &&)

```
if ( nachname.equalsIgnoreCase("Mustermann")  &&  note == 5 )
        {
        System.out.println("das ist aber garantiert ein Tippfehler! " );
        }
```

Freiwillige Zusatzbeispiele für Fortgeschrittene

Ergänzung zu Kontrollstrukturen

Es gibt neben dem `if` noch weitere Kontrollstrukturen für Verzweigungen: `switch`




```
public class Notenskala {

  public static void main(String[] args) {


    int note;
    note = 1;

    System.out.print("Note ");
    System.out.print(note );
    System.out.print(" bedeutet ");

    String note_im_klartext;

    switch (note) {      // switch funktioniert in Java nur mit int und char!
    case 1:
         note_im_klartext="Sehr gut";
         break;                  // ohne break würde der nächste case abgearbeitet
    case 2:
         note_im_klartext="Gut";
         break;
    case 3:
         note_im_klartext="Befriedigend";
         break;
    case 4:
         note_im_klartext="Genuegend";
         break;
    case 5:
         note_im_klartext="Nicht genuegend";
         break;
    default :
         note_im_klartext="--ungueltig--";
    }

    System.out.println(note_im_klartext );

  }
}
```

Beispiel (39): Schreiben Sie ein Programm, das aus einer Zahl einen ausgeschriebenen Wochentag macht (0 → Montag, 1 → Dienstag, …, 6 → Sonntag)

Beispiel (40): Mit folgender Formel können Sie die Wochentagszahl für ein gegebenes Datum berechnen. Die Ausgabe des Wochentags soll in ausgeschriebener Form erfolgen.

```
    int j,m,t,n,wochentag;

    j=2005;
    m=4;
    t=1;

    if (m>=3) {m=m+1;} else {m=13+m; j=j-1;}
    n=(int)(365.25*j)+(int)(30.6*m)+t-621049;
    n=n-1;
    wochentag=n-7*(int)(n/7);
    System.out.println(wochentag);
```

Beispiel (41): Vorgriff auf Schleifen: Schreiben Sie ein Programm, das in einer Schleife alle Wochentage ausgibt.

Schleifen, wiederholte Ausführung

Eine Schleife besteht aus einer **Bedingung** (einem Ausdruck, der einen boole'schen Wert ergibt) und einem **Anweisungsblock**. Der Anweisungsblock (in Java durch geschwungene Klammern **{…}** zusammengefasst) wird solange wiederholt, solange die Bedingung zutrifft (`true` ergibt). Der/die Programmierer/in muss dafür sorgen, dass die Bedingung irgendwann einmal auch `false` ergibt, sonst wird die Schleife niemals verlassen (Endlosschleife).

for (Initialisierung der Laufvariablen; Bedingung; Erhöhung/Erniedrigung der Laufvariablen)
{ Anweisungsblock, der wiederholt wird }
typischer Anwendungsfall für for-Schleifen sind Zählschleifen (Anzahl der Durchläufe bekannt)

while (Bedingung) { Anweisungsblock, der wiederholt wird }
allgemeinste Form der Schleife, wird 0 bis ∞ mal durchlaufen

```
while ( Bedingung )
      {
      Anweisungsblock;
      }
```

do { Anweisungsblock, der wiederholt wird } **while** (Bedingung)
Bedingung wird erst am Ende überprüft, Anweisungsblock wird mindestens 1mal durchlaufen; typischer Anwendungsfall ist das Warten auf eine bestimmte Benutzereingabe.

Initialisierung der Laufvariablen (Startwert, i wird mit dem Wert 1 initialisiert)

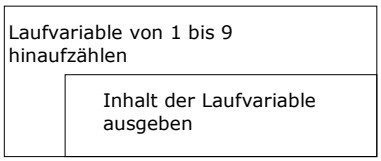
Bedingung: Schleife wird solange ausgeführt, solange die Bedingung `true` ergibt

Erhöhung des Wertes der Laufvariablen

```
for (int i=1; i<10; i=i+1)
    {
     System.out.println(i);
    }
```

Obiges Programm gibt die Zahlen 1 bis 9 aus

Struktogramm:

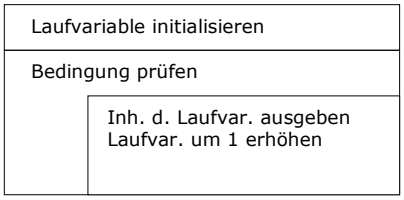


Die beiden untenstehenden Programme machen genau das gleiche:

```
i=1;  // Laufvariable wird vorher initialisiert
while ( i<10 )
      {
       System.out.println(i);
       // nicht vergessen, die Laufvar. zu erhöhen!!
       i=i+1;
      }
```

Die Schleife wird solange ausgeführt, solange i kleiner 10 ist.

Struktogramm:

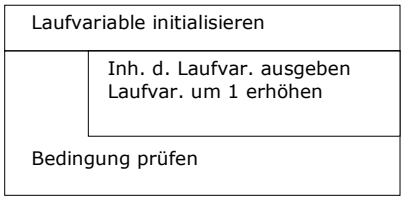


Die do-Schleife wird mindestens 1mal durchlaufen, da die Überprüfung der Bedingung erst am Ende erfolgt.

```
i = 1;
do {
    System.out.println(i);
    i=i+1;
   }
   while ( i<10 );
```

Die Schleife wird solange ausgeführt, solange i kleiner 10 ist.

Struktogramm:



Beispiel (42): Probieren Sie alle drei Schleifentypen aus, um die Zahlen von 1 bis 15 auszugeben.

Übungen (wählen Sie den jeweils geeignetsten Schleifentyp):

Beispiel (43): Schreiben Sie ein Prgrämmchen, das nur die ungeraden Zahlen von 3 bis 15 ausgibt
3 5 7 9 11 13 15
(indem Sie den Anfangswert und die Schrittweite anpassen).

Beispiel (44): Schreiben Sie ein Prgrämmchen, das die Zahlen von 5 bis -5 ausgibt (indem Sie von der Zählvariable subtrahieren statt zu addieren). 5 4 3 2 1 0 -1 -2 -3 -4 -5

Beispiel (45): Schreiben Sie ein Prográmmchen, das eine Liste der Zweierpotenzen von 0 bis 10 erstellt (siehe Beispiel (12): Seite 8).

Entwickeln Sie das Programm in kleinen Schritten!

- schreiben Sie zuerst eine Schleife, die von 1 bis 10 zählt, und sobald das funktioniert,
- geben Sie zusätzlich zur Laufvariablen auch noch 2 hoch Laufvariable aus

```
2 hoch 0 ist 1.0
2 hoch 1 ist 2.0
2 hoch 2 ist 4.0
2 hoch 3 ist 8.0
2 hoch 4 ist 16.0
2 hoch 5 ist 32.0
2 hoch 6 ist 64.0
2 hoch 7 ist 128.0
2 hoch 8 ist 256.0
2 hoch 9 ist 512.0
2 hoch 10 ist 1024.0
```

Beispiel (46): Schreiben Sie ein Prográmmchen, das die Zahlen 0 bis 32 in binärer Darstellung ausgibt (siehe Beispiel (22): Seite 11).
Schreiben Sie dazu eine Schleife, die mit Hilfe der Laufvariablen i die Zahlen 0 bis 32 erzeugt, und geben Sie in der Schleife die jeweils zugehörige Binärzahl aus: `System.out.println(Integer.toBinaryString(i));.`

```
0
1
10
11
100
101
110
111
1000
```

Beispiel (47): Erzeugen Sie 20 ganzzahlige Zufallszahlen zwischen 0 und 99.

Entwickeln Sie das Programm in kleinen Schritten!

- Erzeugen Sie zuerst eine einzelne Zufallszahl im Bereich von 0 bis 99, und sobald das funktioniert,
- schreiben Sie wie gelernt eine Schleife drumherum, die von 1 bis 20 zählt.

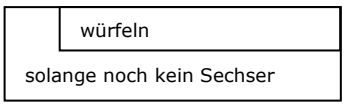
oder:

- schreiben Sie zuerst eine Schleife, die von 1 bis 20 zählt, und sobald das funktioniert,
- geben Sie statt der Laufvariablen eine Zufallszahl aus

Beispiel (48): Schreiben Sie ein Programm, das die Summe aller Zahlen von 1 bis 100 berechnet.
1 + 2 + 3 + 4 + 5 + … + 99 + 100 → 5050
Schreiben Sie dazu eine Schleife, die die Zahlen 1 bis 100 erzeugt. Weiters benötigen Sie eine Variable (z.B. `summe`), in die Sie nacheinander alle erzeugten Zahlen hineinsummieren, z.B. `summe=summe+i`.

Beispiel (49): Schreiben Sie ein Programm, das die Fakultät (Faktorielle) von 7 berechnet (Produkt aller Zahlen von 1 bis 7). Das funktioniert völlig analog zum vorigen Beispiel, nur mit Produkt statt Summe. 5040

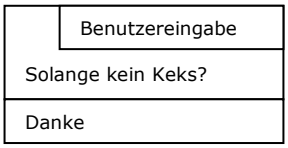
Beispiel (50): Ein typischer Anwendungsfall für eine do-Schleife:
Würfeln Sie so lange, bis Sie einen Sechser gewürfelt haben!
Beachten Sie die Struktogrammdarstellung für diesen Schleifentyp: Der Anweisungsblock wird (mindestens 1x) ausgeführt, und erst danach wird die Bedingung überprüft.



Beispiel (51): Ein typischer Anwendungsfall für eine do-Schleife:
Vom Benutzer wird eine Eingabe gefordert, und die Schleife so lange durchlaufen, bis der Benutzer "Keks" eingibt.
Beachten Sie die Struktogrammdarstellung für diesen Schleifentyp: Der Anweisungsblock wird mindestens 1x ausgeführt, und erst danach wird die Bedingung überprüft.
Bezüglich Verneinen mit ! und .equalsIgnoreCase siehe Seite 16.

```
String antwort;
do {
    System.out.print("Gib mir ein Keks: ");
    antwort = Einlesen.String();
   }
   while ( !antwort.equalsIgnoreCase("Keks") );
System.out.println("Danke!");
```

```
Gib mir ein Keks: Banane
Gib mir ein Keks: Butterbrot
Gib mir ein Keks: Keks
Danke!
```



Beispiel (52): Das vorige Beispiel wird ausgebaut: Das Programm erklärt dem Benutzer sehr deutlich, was es nicht haben möchte:

```
Gib mir ein Keks: Brot
Ich will aber kein Brot, ich will ein Keks!
Gib mir ein Keks: Zuckerl
Ich will aber kein Zuckerl, ich will ein Keks!
Gib mir ein Keks: Keks
Danke!
```

```
String antwort;
boolean fertig = false;
do {
    System.out.print("Gib mir ein Keks: ");
    antwort = Einlesen.String();
    if ( antwort.equalsIgnoreCase("Keks") )
            {
             System.out.println("Danke!");
             fertig = true;
            }
        else
            {
             System.out.print("Ich will aber kein " + antwort);
             System.out.println(", ich will ein Keks!");
            }
   }
   while ( !fertig );
```

Wir verwenden hier eine Variable namens `fertig` vom Typ `boolean`. Solange diese Variable den Inhalt `false` hat ("nicht fertig"), wird die Schleife wiederholt.
Erst wenn der Benutzer ein "Keks" eingibt, wird die Variable `fertig` auf `true` gesetzt, und damit die Schleife verlassen.

Für die folgenden Beispiele gilt: Machen Sie sich *zuerst* eine Skizze vom String, schreiben Sie auf, welche Indizes verwendet werden müssen. Entwickeln Sie das Programm *in kleinen Schritten*! Lassen Sie sich zunächst die Indizes anzeigen, bevor Sie versuchen, Zeichen auszuschneiden!

Beispiel (53): Zerlegen Sie einen String in einzelne Zeichen: Sehen Sie auf Seite 12 nach, wie Sie ein Zeichen an einer bestimmten Position bekommen. Sehen Sie weiters nach, wie man die Länge eines String bestimmt, und entwickeln Sie eine Schleife, die nacheinander alle Zeichen des Strings, jedes in einer neuen Zeile, ausgibt.

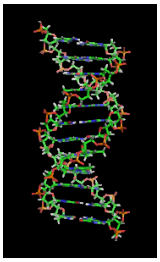
Beispiel (54): Schreiben Sie ein Programm, das einen String umgedreht wieder ausgibt: "Verkehrt" → "trhekreV"

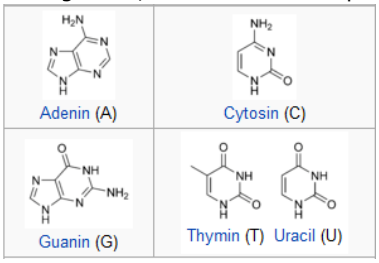
Es gibt wie immer verschiedene Möglichkeiten, diese Aufgabe zu lösen. Entscheiden Sie sich, wie die Indices aussehen sollten: Vom letzten bis 0 heruntergezählt, oder von 0 bis zum letzten Zeichen hinaufgezählt. Lassen Sie zuerst nur die Indices anzeigen, und erst wenn Sie sicher sind, dass Sie die richtigen Zahlen bekommen, holen Sie sich auch die entsprechenden Zeichen aus dem String.

Beispiel (55): Gute Passwörter sollten NICHT aus Ihrem persönlichen Umfeld stammen (Name der Freundin, Lieblingsmotorradmarke, Figur aus Herr der Ringe, …), und in keinem Wörterbuch vorkommen (Passwortknackprogramme arbeiten mit Wörterbüchern), mindestens 8 Zeichen lang sein und auch Ziffern enthalten.

Entwickeln Sie einen **Passwortgenerator**, der Ihnen Vorschläge für Passwörter macht, die nur aus zufälligen Zeichen bestehen:
- Erstellen Sie einen String mit allen erlaubten Zeichen,
  z.B. `String erlaubtezeichen="abcdefghijklmnopqrstuvwxyz+-/*~#_,.;:0123456789";`
- Dann soll per Zufallszahl ein einzelnes Zeichen aus diesem String ausgewählt werden.
- Sobald dieser Teil perfekt funktioniert, können Sie eine Schleife drumherum schreiben, die das achtmal ausführt.

Beispiel (56): Schreiben Sie ein Programm, das eine Basensequenz (der Basen **T**hymin, **A**denin, **C**ytosin, **G**uanin) wie "catggctgcagca…" in 3 Basen lange Stücke (Tripletts, Codons) zerlegt und ausgibt, jedes Stück in einer neuen Zeile. (Wir nehmen zur Vereinfachung nur Strings mit einer durch 3 teilbaren Länge.)

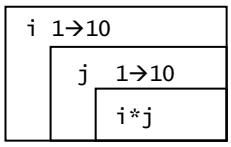





```
Basensequenz: catggctgcagca

Tripletts:
cat
ggc
tgc
agca
```

Quelle: http://de.wikipedia.org/wiki/Nukleobasen

Beispiel (57): Ineinander geschachtelte Schleifen: Schreiben Sie ein Programm, das das kleine Einmaleins (1*1 bis 10*10) ausgibt: 1 mal 1 ist 1, 1 mal 2 ist 2, …
Lösen Sie das mit ineinander geschachtelten Schleifen: eine äußere Schleife, die i von 1 bis 10 hinaufzählt, und eine innere Schleife, die j von 1 bis 10 hinaufzählt; dann brauchen Sie nur mehr i*j berechnen und ausgeben, Wichtig: Zeichnen Sie *zuerst* ein Struktogramm.



Die Ausgabe des Einmaleinsprogramms nur zu einem kleinen Teil am Bildschirm sichtbar bleibt, was selbstverständlich nicht befriedigend ist. Daher sollte die Ausgabe in eine Textdatei geschrieben werden, damit man sie sich in Ruhe im Editor anschauen bzw. in andere Programme übernehmen kann.

Dabei hat man 2 Möglichkeiten:

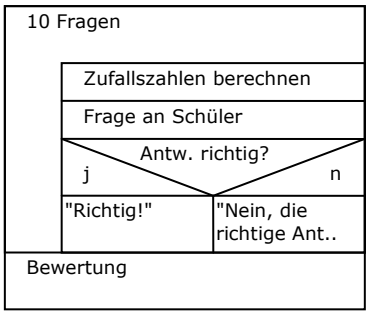
Explizites Schreiben in eine Ausgabedatei aus dem Java-Programm heraus, siehe später Seite 29

Wir haben uns letztes Semester schon mit der Ausgabeumleitung in der Kommandozeile beschäftigt: Wie Sie sich sicherlich erinnern, haben wir die Ausgabe des `dir`-Befehls folgendermaßen in eine Textdatei umgeleitet: `dir > inhalt.txt`
Das können wir mit unseren Java-Programmen natürlich auch machen, z.B.:

Probieren Sie es im schwarzen Fenster aus!

```
U:\EinfDV\JavaKurs>  java Einmaleins > Einmaleins.txt
U:\EinfDV\JavaKurs>  notepad  Einmaleins.txt
```

Beispiel (58): Schreiben Sie ein Rechentrainingsprogramm für Volksschüler:
1. Das Programm denkt sich 2 ganzzahlige Zufallszahlen aus, und fragt den Schüler nach der Summe (z.B. "3+5=?").
2. Der Schüler gibt die Summe, die er im Kopf berechnet hat, ein.
3. Das Programm vergleicht die Eingabe mit der richtigen Summe, und gibt einen Kommentar aus. Gleichzeitig werden richtige und falsche Antworten gezählt.
4. In einer Schleife werden Punkte 1.-3. 10mal wiederholt
5. Das Programm gibt die Anzahl der richtigen Antworten aus



Basensequenz → Aminosäure (Zusatzbeispiel für Fortgeschrittene)

Beispiel (59): Entwickeln Sie ein Programm, das einen String mit **Basensequenzen** (z.B. "cagttacgatagc..") in Tripletts zerlegt, und die entsprechende **Aminosäuresequenz** ausgibt.

Standard-Codon-Tabelle. Diese Tabelle zeigt die 64 möglichen Basen-Tripletts.

1. Base	2. Base U	2. Base C	2. Base A	2. Base G
U	UUU Phenylalanin	UCU Serin	UAU Tyrosin	UGU Cystein
	UUC Phenylalanin	UCC Serin	UAC Tyrosin	UGC Cystein
	UUA Leucin	UCA Serin	UAA *Stop*	UGA *Stop*
	UUG Leucin	UCG Serin	UAG *Stop*	UGG Tryptophan
C	CUU Leucin	CCU Prolin	CAU Histidin	CGU Arginin
	CUC Leucin	CCC Prolin	CAC Histidin	CGC Arginin
	CUA Leucin	CCA Prolin	CAA Glutamin	CGA Arginin
	CUG Leucin	CCG Prolin	CAG Glutamin	CGG Arginin
A	AUU Isoleucin	ACU Threonin	AAU Asparagin	AGU Serin
	AUC Isoleucin	ACC Threonin	AAC Asparagin	AGC Serin
	AUA Isoleucin	ACA Threonin	AAA Lysin	AGA Arginin
	AUG Methionin*	ACG Threonin	AAG Lysin	AGG Arginin
G	GUU Valin	GCU Alanin	GAU Asparaginsäure	GGU Glycin
	GUC Valin	GCC Alanin	GAC Asparaginsäure	GGC Glycin
	GUA Valin	GCA Alanin	GAA Glutaminsäure	GGA Glycin
	GUG Valin	GCG Alanin	GAG Glutaminsäure	GGG Glycin

Quelle: http://de.wikipedia.org/wiki/Codon

(nicht eintippen, gibt es in `JavaKurs\Codons.java`)

```
public class Codons {

  public static void main(String[] args) {

  String c="tgg";

  c=c.toUpperCase();       // alles einheitlich auf GROSS
  c=c.replace('U', 'T');   // Uracil --> Thymin
  String asn="ungueltig", asc="---";     // Aminosäurename, AS-Code
  if(c.equals("GCA") || c.equals("GCC") || c.equals("GCG") ||
c.equals("GCT")){asn="Alanin"; asc="Ala";}
  if(c.equals("CTA") || c.equals("CTA") || c.equals("CTG") || c.equals("CTT") ||
c.equals("TTA") || c.equals("TTG")){asn="Leucin"; asc="Leu";}
  if(c.equals("GGA") || c.equals("GGC") || c.equals("GGG") ||
c.equals("GGT")){asn="Glycin"; asc="Gly";}
  if(c.equals("GTA") || c.equals("GTC") || c.equals("GTG") ||
c.equals("GTT")){asn="Valin"; asc="Val";}
… … …
… … …
… … …
```

Beispiel (60): Vorgriff auf Ein-/Ausgabe: Entwickeln Sie – Schritt für Schritt – ein Programm, das aus einer Textdatei Basensequenzen einliest, den Teil zwischen Start- und Stopp-Codons bestimmt, und die Übersetzung in die entsprechenden Aminosäuren durchführt.

Kryptographie (Zusatzbeispiel für Fortgeschrittene)

Die Kryptographie beschäftig sich mit der Verschlüsselung von Informationen ("Geheimschriften").
(Für Interessierte kann das sehr spannend geschriebene Buch von Simon SINGH empfohlen werden: Geheime Botschaften. Die Kunst der Verschlüsselung von der Antike bis in die Zeiten des Internet. dtv 2001, ISBN 3-423-33071-6)

Cäsar-Verschlüsselung

G.J.Cäsar verwendete im Gallischen Krieg folgende einfache Verschlüsselung:
Das Alphabet wurde um n Stellen verschoben.

Beispiel: n=3, aus GEHEIM wird DBEBFJ

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
        ↓   ↓ ↓ ↓       ↓
→ → → A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

Die Entschlüsselung erfolgt mit dem gleichen Schlüssel, nur mit umgekehrten Vorzeichen.

Bei der originalen Cäsar-Verschlüsselung wurden die Buchstaben, die am Ende herausragen, am Beginn eingefügt:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
        ↓   ↓ ↓ ↓       ↓
X Y Z A B C D E F G H I J K L M N O P Q R S T U V W
```

Sie können dieses Verhalten mittels Modulo-Arithmetik (siehe S.11) implementieren, oder ganz einfach mit 2 Strings, und der zweite ist eben um n Zeichen im Kreis verschoben:

```
String s1="ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜ0123456789 .,!?"
                ↓
String s2="HIJKLMNOPQRSTUVWXYZÄÖÜ0123456789 .,!?ABCDEFG"

String s3="SECRET"
                ↓
String s4="ZLJYLÄ"
```

Bilden Sie Teams, und diskutieren Sie Datenstrukturen und Algorithmen, die sich eignen, um eine Cäsar-Verschlüsselung zu implementieren.

Beispiel (61): Erstellen Sie im Team ein Programm, das eine Zeile Text ver- bzw. entschlüsselt.

Beispiel (62): Tauschen Sie verschlüsselte Nachrichten (Spielregel: sinnvoller Text in deutscher Sprache) mit anderen Teams aus, diese sollten die Nachricht entschlüsseln können, sobald Sie Ihnen den Schlüssel verraten.

Beispiel (63): Die ganz Fortgeschrittenen unter Ihnen dürfen sich an einem **Codebrecher**-Programm versuchen …

Das könnte man mit einer sogenannten **Brute-Force-Attacke** versuchen: darunter versteht man das Durchprobieren sämtlicher Schlüssel. In unserem Fall ist das relativ leicht, es gibt nicht sehr viele mögliche Schlüssel, und Sie können selbst sehr leicht ein Programm schreiben, das in einer Schleife Schlüssel durchprobiert. Sie sehen dann am Bildschirm (oder in der Ausgabedatei) sofort, wenn ein sinnvoller Text herausgekommen ist.

Sie können die Verschlüsselung sicherer machen, indem Sie die Zeichen des String s2 nicht um einen festen Wert verschieben (denn dann ist dieser Wert der Schlüssel, und es gibt nicht allzu viele Möglichkeiten), sondern den String s2 völlig durcheinanderwürfeln, und dann den String s2 als Schlüssel verwenden. In diesem Fall gibt es eine wesentlich größere Zahl von Schlüsseln.

Ein weiterer Ansatz, solche monoalphabetischen Substitutionen zu brechen, ist die **Häufigkeits-analyse**. In der deutschen Sprache kommt der Buchstaben 'e' mit Abstand am häufigsten vor. Sie brauchen in einem verschlüsselten Text also nur das häufigste Zeichen zu suchen, und Sie haben mit großer Sicherheit die Codierung des 'e' gefunden.
Weitere Buchstabenhäufigkeiten: http://de.wikipedia.org/wiki/Buchstabenhäufigkeit

Arrays (Felder, indizierte Variable)

Ein Array ist eine Variable, die aus vielen durchnummerierten Elementen vom gleichen Datentyp besteht.
Da die Elemente durchnummeriert sind, können die Elemente eines Arrays mit Schleifen abgearbeitet werden (statt a+b+c+… → $a_1+a_2+a_3+\ldots$). Die Indices beginnen bei Null.

```
String[] obst = new String[4];      // Deklaration eines Arrays mit 4 Elementen
obst[0] = "Apfel";
obst[1] = "Birne";
obst[2] = "Zwetschke";
obst[3] = "Marille";

System.out.println( obst[2] );  // Element 2 des Arrays obst
```

Mac OS X: eckige Klammern: [mit ⌥ + 5,] mit ⌥ + 6

Beispiel (64): Probieren Sie obiges Beispiel mit verschiedenen Indexwerten aus.
Was passiert bei Indexwert 9 ?

Beispiel (65): Schreiben Sie ein Prograämmchen, das in einer Schleife alle Elemente des `obst[]`-Arrays ausgibt. `obst.length` (ohne Klammern) liefert die Größe des Arrays (Anzahl der Elemente; die Indizes laufen aber wie bei den Strings von 0 bis Länge minus 1).

Beispiel (66): Menü

In folgendem Beispiel nimmt das Array die Speisen einer Speisekarte auf.
Das Programm soll folgende Aufgaben erfüllen:

1. die Speisekarte anzeigen
2. den Benutzer nach einer Auswahl fragen
3. prüfen, ob die Auswahl im gültigen Bereich ist, und falls nicht, Benutzer noch einmal fragen, so oft, bis eine gültige Eingabe erfolgt (der Einfachheit halber tippen wir nur numerische Werte ein)
4. ausgewählte Speise anzeigen

```
============================
         Speisekarte
============================
1.) Burger
2.) Wuerschtl
3.) Eintopf
4.) Nudeln
----------------------------
Bitte waehlen Sie aus: 6
Ihre Auswahl ist ungueltig,
bitte waehlen Sie Werte zwischen 1 und 4
----------------------------
Bitte waehlen Sie aus: 1

Sie haben Burger gewaehlt, kommt sofort.
```

```
public class Speisekarte {

    public static void main(String[] args) {


    String[] speise = new String[4];
    speise[0]="Burger";
    speise[1]="Wuerschtl";
    speise[2]="Eintopf";
    speise[3]="Nudeln";

    System.out.println("============================");
    System.out.println("         Speisekarte");
    System.out.println("============================");
    for (int i=0; i<speise.length; i++) {
        System.out.println((i+1) + ".) " + speise[i]);
        }
    int auswahl;
    System.out.println("----------------------------");
    System.out.print("Bitte waehlen Sie aus: ");
    auswahl=Einlesen.Int();
    System.out.println();
    System.out.println("Sie haben " + speise[auswahl-1] + " gewaehlt, kommt sofort." );
    }
}
```

Beispiel (67): **Arithmetisches Mittel, Standardabweichung**

In folgendem Beispiel dient ein Array zur Aufnahme von Zahlenwerten (die z.B. über Tastatur oder aus einer Datei eingelesen werden könnten). In der ersten Schleife über alle Array-Elemente werden Anzahl und Summe ermittelt. In der zweiten Schleife wird dann die Summe der quadrierten Abweichungen der Einzelwerte vom Mittelwert berechnet.

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

```
public class Statistik2 {
    public static void main(String[] args) {

    double[] wert = new double[4];
    wert[0]=27.3;
    wert[1]=19.5;
    wert[2]=38.5;
    wert[3]=21.8;

    double summe=0; int anzahl=0;
    for (int i=0; i<wert.length; i++) {
          System.out.println((i+1) + ".Wert: " + wert[i]);
          summe=summe+wert[i];
          anzahl=anzahl+1;
        }

    System.out.println("---------------------");
    System.out.println("Summe:      " + summe );
    System.out.println("Anzahl:     " + anzahl );
    double mittelw=summe/anzahl;
    System.out.println("Mittelwert: " + mittelw );

    double abweichungsquadratsumme=0;
    for (int i=0; i<wert.length; i++) {
          abweichungsquadratsumme=abweichungsquadratsumme+Math.pow(wert[i]-mittelw,2);
        }
    System.out.println("Summe der quadrierten Abweichungen: " + abweichungsquadratsumme);
    System.out.println("Varianz: " + abweichungsquadratsumme/(anzahl-1) );

    }
}
```

```
1.Wert: 27.3
2.Wert: 19.5
3.Wert: 38.5
4.Wert: 21.8
---------------------
Summe:      107.1
Anzahl:     4
Mittelwert: 26.775
Summe der quadrierten
Abweichungen: 215.4275
Varianz: 71.80916666666667
```

Sortieralgorithmen (Zusatzbeispiele für Fortgeschrittene)

Sortieralgorithmen zählen zu den Grundalgorithmen in der Informatik, und es gibt zahllose davon. Falls Sie sich für die Vertiefungsrichtung Bioinformatik interessieren, sollten Sie sich näher damit befassen. Außerdem handelt es sich um eine gute Übung:

Beispiel (68): Sortier-Algorithmus 1

Sie vergleichen das 1. und das 2. Element. Ist der Wert des 2. Elements kleiner als der Wert des 1., vertauschen Sie die Inhalte des 1. und des 2. Elements.
Sie vergleichen das 2. und das 3. Element. Ist der Wert des 3. Elements kleiner als der Wert des 2., vertauschen Sie die Inhalte des 2. und des 3. Elements.
….
Sie vergleichen das vorletzte (n-1) und das letzte (n) Element. Ist der Wert des letzten Elements kleiner als der Wert des vorletzten, vertauschen Sie die Inhalte des vorletzten und des letzten Elements.

Wiederholen Sie den Vorgang mit der ganzen Liste. Falls bei einem solchen Durchgang keine Vertauschung mehr stattfindet, ist die Liste fertig sortiert.

Beispiel (69): Sortier-Algorithmus 2

Zeigen Sie auf das 1. Element, und durchsuchen Sie den Rest der Liste, ob Sie ein noch kleineres finden. Wenn ja, vertauschen Sie die Inhalte. Wenn Sie die Liste durch sind, steht der kleinste Wert sicher ganz oben.

Zeigen Sie nun auf das 2. Element, und durchsuchen Sie den Rest der Liste, ob Sie ein noch kleineres finden. Wenn ja, vertauschen Sie die Inhalte. Wenn Sie die Liste durch sind, steht der zweitkleinste Wert an Stelle 2.

Weiter so bis zum vorletzten Element.

Ein- / Ausgabe (Input / Output, IO)

Java bietet hier eine relativ unübersichtliche Vielfalt an Möglichkeiten, Datenströme (streams) einzulesen. Für unseren Kurs reicht es, folgendes Grundgerüst für das Einlesen einer Zeile einer einfachen Textdatei als Kochrezept zu verwenden:

```
import java.io.*;

public class DateiLesen01{

	public static void main(String[] args) throws IOException {

	BufferedReader r1 = new BufferedReader( new FileReader("Textdatei1.txt") );
		String zeile;

		zeile = r1.readLine();  // nächste Zeile vom Reader r1 einlesen
		System.out.println(zeile );

	r1.close();
	}
}
```

Unter einer einfachen Textdatei verstehen wir eine Datei, die man mit einem Editor erstellt bzw. bearbeitet. Das wollen wir gleich tun: Schreiben Sie mittels Editor 3 Zeilen beliebigen Text, und speichern Sie diesen im gleichen Ordner wie Ihre Java-Programme unter dem Namen "Textdatei1.txt".

Neu in diesem Beispiel:

- `import java.io.*` Zur Verwendung von IO-Funktionalität (IO … Input/Output) muss das Package java.io importiert werden
  Falls wir das vergessen, schreit der Compiler natürlich: `cannot find symbol` bzw. `Error: Type "BufferedReader" was not found.`
- `throws IOException` Behandlung von Fehlern und Ausnahmen (exceptions) wird später in der Vorlesung behandelt, bei Verwendung von Ein-/Ausgabe ist dieser Zusatz aber notwendig.
- `FileReader, BufferedReader` Die angegebene Textdatei wird geöffnet, die Variable `r1` (Name beliebig wählbar) enthält alle für Java notwendigen technischen Details (vergleiche file handle in anderen Programmiersprachen), daher wichtig: bei späterer Bezugnahme im Programm auf diese geöffnete Datei verwenden wir `r1`, nicht den Dateinamen.
  (Detaillierte Erklärungen bitte in der Literatur nachlesen)
- `.readLine()` Die Methode `readLine()` liest die nächste Zeile aus der Textdatei und gibt diese als String zurück. Da wir diesen String weiterverarbeiten möchten, speichern wir ihn in der Variable zeile, die wir zuvor als String deklariert haben.
- `.close()` Die Methode `close()` schließt die Datei.

Was macht nun dieses Beispiel: Die angegebene Textdatei wird geöffnet, die erste Zeile eingelesen, und am Bildschirm ausgegeben, fertig, nichts weiter.

Das ist schon einmal ein guter Anfang, aber insofern unbefriedigend, als wir wissen, dass unsere Textdatei mehr als eine Zeile enthält. Wir probieren daher folgendes: Wir kopieren den Code für Zeile Einlesen und Ausgeben mehrfach.

```
zeile = r1.readLine();
System.out.println(zeile );
zeile = r1.readLine();
System.out.println(zeile );
zeile = r1.readLine();
System.out.println(zeile );
zeile = r1.readLine();
System.out.println(zeile );
```

Probieren Sie es aus. Es funktioniert. Das bringt uns eine wichtige Erkenntnis: Das Einlesen von einer Textdatei funktioniert, indem wir uns Schritt für Schritt Zeile für Zeile holen. **Jeder Aufruf von `readline()` liefert uns die nächste Zeile aus der Datei**, die wir dann beliebig weiterverarbeiten können.

Was an dieser Variante unbefriedigend ist, liegt auf der Hand: Eine Textdatei kann hunderte, tausende Zeilen haben, und es ist nicht sinnvoll, die Anweisungen so oft zu kopieren. Also: wie immer bei Dingen, die wiederholt ausgeführt werden: eine Schleife muss her!
Die Frage ist nur: Was verwenden wir als Abbruchbedingung für die Schleife? Die Schleife soll so lange ausgeführt werden, bis alle Zeilen abgearbeitet sind, also keine Zeilen mehr übrig sind. Wie formulieren wir diese Bedingung?
Sehen Sie sich dazu die Ausgabe des letzten Beispiels an: Falls Sie tatsächlich 3 Zeilen in der Textdatei haben, und 5mal `r1.readline()` aufrufen, bekommen Sie Ihre 3 Textzeilen auf den Bildschirm, und dann 2mal `null`. Das sollte Sie als Programmierer/in auf folgende Idee bringen: Wir lesen solange ein, bis wir statt eines Strings einen null-Wert zurückbekommen!

So ähnlich würde es funktionieren:
zuerst ein `zeile = r1.readLine()`,
dann ein `if (zeile != null ){` Zeile verarbeiten `} else {` Schleife beenden `}`
Das passt aber nicht ganz in unser Schleifenkonzept: Wir wünschen uns einen Ausdruck im Bedingungsteil der while-Schleife. Wir machen uns daher zunutze, dass eine Wertzuweisung an eine Variable einen Wert zurückliefert, den wir unmittelbar für einen Vergleich nützen können, was zu folgendem kompakten Ausdruck führt:

```
while (  ( zeile = r1.readLine() ) != null  ) { Zeile verarbeiten }
```

Das führt uns zu folgendem ultimativen **Grundgerüst für das Einlesen von Textdateien**:

```
import java.io.*;

public class DateiLesen01c{

	public static void main(String[] args) throws IOException {
		BufferedReader r1 = new BufferedReader( new FileReader("Textdatei1.txt") );
		String zeile;

		while ( (zeile = r1.readLine()) != null ) {
			// zeilen verarbeiten
			System.out.println(zeile );
		}
	r1.close();
	}
}
```

Probieren Sie es aus!

Sobald das Programm funktioniert erweitern Sie es zur Übung um folgende Funktionalität:
Vor jeder Zeile soll eine Zeilennummer ausgegeben werden. Überlegen Sie, das Sie dazu brauchen (vielleicht eine Zählvariable? Von welchem Typ? Wo platziert? Wie erhöht man den Wert einer Variablen? An welcher Stelle im Programm ist das Inkrement sinnvoll? Wie geben Sie die Zeilennummer aus? Wie kann man das noch hübscher machen? …)

Beispiel (70): Im Ordner `JavaKurs` befindet sich eine Datei `Namensliste.txt` (bzw. laden Sie diese Datei aus Moodle herunter.)
Schreiben Sie ein Programm, das diese Namensliste Zeile für Zeile einliest und am Bildschirm ausgibt. Wenn das funktioniert, führen Sie eine Zählvariable für die Zeilenzahl ein, und geben Sie vor jeder Zeile die Zeilennummer am Bildschirm aus.

Beispiel (71): Schreiben Sie ein Programm, das die Zeilen einer Textdatei in ein Array einliest, und dann verkehrt herum (die letzte Zeile zuerst) am Bildschirm wieder ausgibt.

Eingabe-Datenströme (streams) können von einer Datei kommen, aber genauso gut von einer Tastatur oder einer anderen Datenquelle oder einem Eingabegerät. Ausgabe-Datenströme können in eine Datei geleitet werden, aber genauso gut an andere Ausgabegeräte, z.B. Bildschirm. Während man Dateien explizit ansprechen muss, gibt es zwei fix und fertig verwendbare streams: `System.in` und `System.out` (`System.out` kennen wir schon von `System.out.println()`). In einem schwarzen Fenster können wir damit schon Benutzerinteraktion realisieren.

Unser Grundgerüst für das Einlesen einer Zeile Text lässt sich also folgendermaßen zum einem **Grundgerüst für das Einlesen von Eingabe von der Tastatur** modifizieren:

```
import java.io.*;

public class DateiLesen02{

     public static void main(String[] args) throws IOException {

          System.out.print("Bitte was eintippen: ");

          BufferedReader r2 = new BufferedReader ( new InputStreamReader(System.in) );
          String zeile;
          zeile = r2.readLine();

          System.out.println();
          System.out.println("Du hast Folgendes eingetippt: " + zeile);

     }
}
```

Probieren Sie es aus!

Beim Einlesen mit readLine() bekommen wir Strings. Falls wir numerische Werte erwarten, müssen die Strings erst in numerische Werte umgewandelt werdeN:

```
    int j = Integer.parseInt("123");
    double d = Double.parseDouble("123.5");
```

(Erklärungen zu den Klassen groß Integer und Double in der Vorlesung)

Übungen

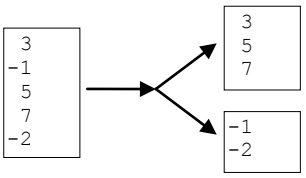
Beispiel (72):	Schreiben Sie ein Programm, das den Benutzer auffordert, eine Zahl einzugeben, und dann alle Zahlen von 1 bis zu dieser Zahl ausgibt.
Beispiel (73):	Schreiben Sie ein Programm, das Zahlen aus einer Textdatei einliest, und den arithmetischen Mittelwert daraus berechnet.
Beispiel (74):	Schreiben Sie ein Programm, das so lange Werte einliest, bis der Benutzer nichts mehr eingibt: zeile.length()=0

Was uns noch fehlt, ist folgendes **Grundgerüst für das Schreiben von Textdateien**:

```
import java.io.*;

public class DateiSchreiben {

  public static void main(String[] args) throws IOException {

    BufferedWriter w1 = new BufferedWriter( new FileWriter( "Ausgabedatei.txt" ));

    w1.write("Blabla");
    w1.newLine();       // Zeilenumbruch
    w1.write("das ist Zeile 2");
    w1.close();

  }
}
```

Übungen:

Beispiel (75):	Experimentieren Sie mit obigem Grundgerüst. Z.B. ist sehr unschön, dass das Programm nicht sagt, was es tut: Der Benutzer soll informiert werden, in welche Datei geschrieben wird.
Beispiel (76):	Schreiben Sie Ihren eigenen copy-Befehl: von einer Textdatei soll gelesen werden, und in eine andere Datei geschrieben werden. Das Programm soll den Benutzer auch informieren, wie viele Zeilen kopiert wurden.
Beispiel (77):	Schreiben Sie ein Programm, wo ein Benutzer Textzeilen eingeben kann, die dann in eine Datei geschrieben werden. Denken Sie über eine Abbruchbedingung nach, d.h. was muss der Benutzer eingeben, damit das Programm aufhört.
Beispiel (78):	Erstellen Sie eine Textdatei mit positiven und negativen Zahlen. Schreiben Sie ein Programm, das diese Datei einliest, die positiven Zahlen in eine Datei 'positiv.txt' schreibt, die negativen Zahlen in eine Datei 'negativ.txt'.



Ausgabe von HTML-Dateien

Wir beschäftigen uns aus folgenden Gründen mit HTML:

1. Es ist möglich, Java-Programme in Webseiten einzubauen: **Applets**
2. Wenn man HTML kann, ist es auf relativ einfache Weise möglich, einigermaßen ansehnlich **formatierte Ausgabedateien** (mit Überschriften, Aufzählungen, Tabellen, ..) zu erzeugen.
3. Wenn Sie verstanden haben, wir man HTML-Ausgabe mit Programmen erzeugt, haben Sie auch das **Prinzip von dynamischen Webseiten** verstanden.

Eine HTML-Datei ist auch nur eine Textdatei, die halt ein bisserl speziellen Text (HTML-Tags) enthält, was dann ein Webbrowser entsprechend interpretiert. Wenn Sie die Tags kennen, kann Sie niemand daran hindern, diese wie normalen Text mit `System.out.println("<html><body>")` oder wie auch immer auszugeben.

Übungen:

Arbeiten Sie zur Wiederholung den Abschnitt HTML im Übungsskriptum durch:
http://statedv.boku.ac.at/roberts_it-kurs-unterlagen/?i=HTML-1

Tabellen:

<tr> .. table row, <td> .. table data

```
<html>
<body>

<table border="1">
<tr> <td>ODER</td>              <td>0</td> <td>1</td>
</tr>
<tr> <td align="right">0</td> <td>0</td> <td>1</td>
</tr>
<tr> <td align="right">1</td> <td>1</td> <td>1</td>
</tr>

</body>
</html>
```

ODER	0	1
0	0	1
1	1	1

Tabellen werden auch oft für Layout-Aufgaben "missbraucht": `<table border="0">` macht eine unsichtbare Tabelle, die sie (eingeschränkt) benützen können, um Elemente (Text, Grafik) am Bildschirm zu platzieren.

Falls Sie eine Ausgabe wie z.B. `<table border="1">` in Java wie gewohnt in doppelte Anführungszeichen einbetten wollen, stoßen Sie natürlich auf das Problem der Anführungszeichen innerhalb von Anführungszeichen: Hier benötigen wir ein sogenanntes Escape-Zeichen, das die Bedeutung von " als Ende eines String-Literals aufhebt:

```
System.out.println( "<table border=\"1\">" );
```

Schaut auf den ersten Blick verwirrend aus: der Backslash wirkt als Escape-Zeichen, der String geht weiter, und beinhaltet die gewünschten Anführungszeichen. Probieren Sie es aus.

Beispiel (79): Bauen Sie das Programm, das die Namensliste einliest, folgendermaßen aus: Die Namen sollen als Aufzählungsliste in eine HTML-Datei ausgegeben werden.

Beispiel (80): Die Namensliste soll als HTML-Tabelle ausgegeben werden, eine Spalte mit Vornamen, eine Spalte mit Nachnamen (selbstverständlich mit Spaltenüberschriften ;-).

Beispiel (81): Verändern Sie das Einmaleins-Programm so, dass die Ausgabe in Tabellenform als HTML-Datei geschrieben wird.

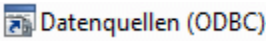
Datenbankanbindung

Im Übungsskriptum finden Sie die Syntax von **SQL** zum Wiederholen:
http://statedv.boku.ac.at/roberts_it-kurs-unterlagen/?i=DB-SQL

ODBC (Open Database Connectivity) ist eine universelle Datenbank-Schnittstelle.

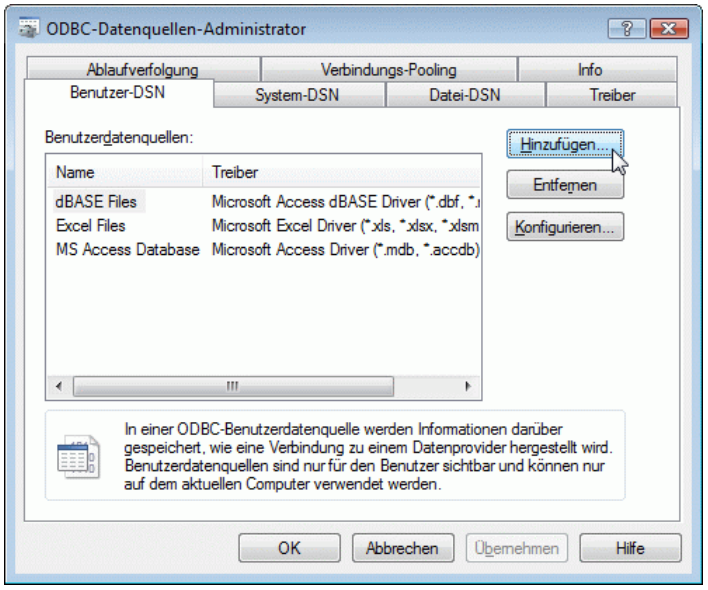
Zunächst müssen wir auf dem PC eine **ODBC-Datenquelle (data source)** einrichten:

'Start' >'Systemsteuerung' > 'Verwaltung' > 'Datenquellen (ODBC)'

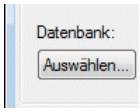


Reiter 'Benutzer-DSN', 'Hinzufügen'

Aus der Liste 'Microsoft Access Driver (*.mdb)' oder 'Microsoft Access Driver (*.mdb, *.accdb)' auswählen, 'Fertig stellen'.

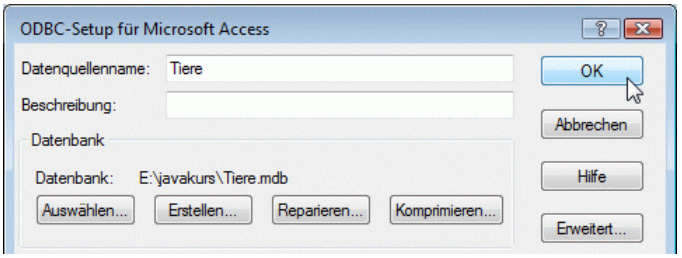


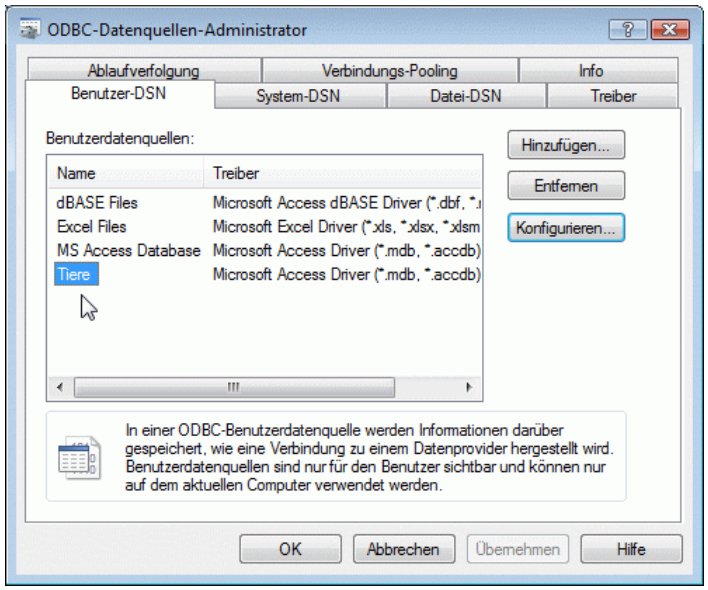
Wir erstellen nun einen Data Source Name (DSN). Wählen Sie dazu eine Datenbank aus: 'Auswählen'.



Im Ordner `JavaKurs\` finden Sie eine MS Access-Datei namens `Tiere.mdb`. Anklicken, 'OK'.

Nun vergeben Sie einen frei wählbaren Datenquellennamen (Data Source Name, DSN), z.B. 'Tiere', mit dem Sie dann auf die Datenbank `JavaKurs\Tiere.mdb` zugreifen





Grundgerüst für den Datenbankzugriff

(nicht abtippen, Datei gibt es in JavaKurs)

```
import java.sql.*;

public class Datenbankbeispiel
{
  public static void main (String[] args)
  {
    try
    {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

      // Aufbau einer Verbindung zur Datenbank
      Connection connection1 = DriverManager.getConnection("jdbc:odbc:Tiere ");
              // oder z.B.   jdbc:odbc:Kursteilnehmer

      Statement statement1 = connection1.createStatement();

      String sql1 = "select * from tiere order by kontinent, name";
      ResultSet resultset1 = statement1.executeQuery(sql1);   // Abfrage wird ausgeführt

      while (resultset1.next())   // nächsten Datensatz holen
      {
        System.out.print( resultset1.getString("kontinent "));
        System.out.print(" " );
        System.out.println( resultset1.getString("name"));
      }

      resultset1.close();
      statement1.close();
      connection1.close();
    }
    catch (Exception problem)   // try .. catch .. dient zur Fehlerbehandlung
    {
      System.out.println(problem.toString());
      System.exit(1);
    }
  }
}
```

Obwohl das Beispiel vielleicht kompliziert aussieht, ist das Grundprinzip doch recht einfach: Wir öffnen eine Verbindung zur einer Datenbank (so wie wir früher eine Eingabedatei geöffnet haben), dann schicken wir ein SQL-Statement an die Datenbank (das ist der große Vorteil der Datenbank gegenüber einer Textdatei, dass hier wesentliche Flexibilität bei der Auswahl der Datensätze besteht), die Datenbank liefert eine Ergebnismenge (result set), das wir wie gewohnt abarbeiten, indem wir mit einer while-Schleife Datensatz für Datensatz abholen.

Beispiel (82): Schreiben Sie ein Programm, das die Tabelle 'Kurs' einliest, den Nachnamen (Feld 'nachn') in Großbuchstaben, und den Vornamen (Feld 'vorn') ausgibt. Das Trennen von Vor- und Nachname ersparen Sie sich hier natürlich, das ist ja der große Vorteil bei einer Datenbank!

Beispiel (83): Schreiben Sie ein Programm, das nicht alle Datensätze ausgibt, sondern nur eine vom Benutzer gewählte Teilmenge: Der Benutzer soll z.B. einen Buchstaben eingeben, und das Programm gibt alle Teilnehmer mit diesem Anfangsbuchstaben aus.

Beispiel (84): Schreiben Sie ein Programm, das die Tabelle 'Tiere' einliest, den Kontinent als Überschrift, und die in diesem Kontinent lebenden Tiere eingerückt ausgibt.
(Hinweise: Sie benötigen dazu eine Variable, in der sich das Programm merkt, welcher Kontinent gerade aktuell ist. Wenn im nächsten eingelesenen Datensatz der Kontinent gleich bleibt, gibt man einfach das Tier aus, ansonsten den Kontinent als Überschrift, und träg den Kontinent in die Variable ein, wo sich das Programm den gerade aktuellen Kontinent merkt.

Afrika
- Elefant
- Giraffe
- Löwe
- Nashorn

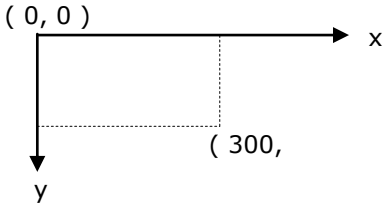
Amerika
- Bison
- Bär

Asien
- Panda

Beispiel (85): Die Tabelle 'Studierende' soll in einer Art Telefonbuch ausgegeben werden, das heißt, der erste vorkommenden Anfangsbuchstabe groß als Überschrift, dann alle Teilnehmer/innen mit Anfangsbuchstaben 'A' eingerückt usw. Das Verfahren ist ganz analog zum vorhergehenden Beispiel.

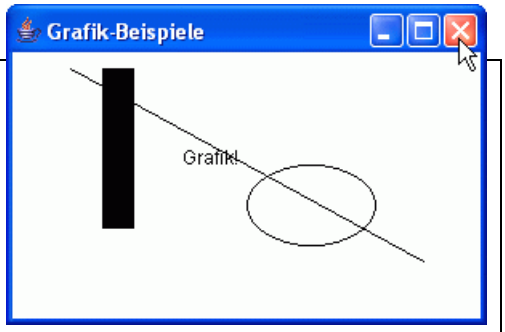
Grafik

Beim Koordinatensystem von Java muss man darauf achten, dass der Ursprung links oben ist.



Grundgerüst eines Grafikprogramms:

(nicht abtippen, Datei gibt es in `JavaKurs`)




```
import java.awt.*;
import java.awt.event.*;

public class Grafikbeispiel extends Frame {

  public void paint (Graphics g){
    g.drawLine(40,40,260, 160);
    g.drawOval(150,100,80,50);
    g.drawString("Grafik!", 110,100);
    g.fillRect(60,40,20,100);
  }


  public Grafikbeispiel (String titel) {
      super(titel);  // blaue Titelleiste des Fensters
      // folgender umständlicher Aufruf ist notwendig,
      // damit sich das Fenster schließen läßt
      addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent evt) { System.exit(0); }
          });
      setSize(600, 400);   // Fenstergöße
      setVisible(true);
      setBackground(Color.white);      //  Hintergrundfarbe des Fensters
  }

  public static void main(String[] args) {
      new Grafikbeispiel("Grafik-Beispiele");
  }
}
```

Für unsere ersten Experimente verwenden wir das Abstract Windowing Toolkit (AWT).

`drawLine(x1, y1, x2, y2)` ..... zeichnet eine Linie vom Punkt1 (x1,y1) zum Punkt2 (x2,y2)

`drawRect(x1, y1, Breite, Höhe)` zeichnet ein Rechteck (rectangle) vom Punkt1 (x1,y1) mit angegebener Breite und Höhe

`fillRect(…)` ..... das gleiche, aber ausgefüllt

`drawOval(x, y, Breite, Höhe)` . zeichnete eine Ellipse mit Mittelpunkt (x,y) und angegebener Breite und Höhe; wenn Breite und Höhe gleich sind, entsteht natürlich ein Kreis

`fillOval(…)` ..... das gleiche, die Ellipse / der Kreis wird aber ausgefüllt

`drawString(Text, x, y)` ..... Der Text wird an angegebener Position in die Grafik eingefügt

Übungen

Experimentieren Sie! Zeichnen Sie was Nettes! (Entwurf auf kariertem Papier kann helfen).

Farben

Erläuterungen zu RGB siehe Übungsskriptum: http://statedv.boku.ac.at/roberts_it-kurs-unterlagen/?i=HTML-1

Standardfarben:

Farbname	RGB-Wert
Color.white	255,255,255
Color.black	0,0,0
Color.lightGray	192,192,192
Color.gray	128,128,128
Color.darkGray	64,64,64
Color.red	255,0,0
Color.green	0,255,0
Color.blue	0,0,255
Color.yellow	255,255,0
Color.magenta	255,0,255
Color.cyan	0,255,255
Color.pink	255,175,175
Color.orange	255,200,0

Benutzerdefinierte Farben:

```
Color meineLieblingsfarbe = new Color(140,140,140);


g.setColor(Color.green);          // alles, was ab jetzt gezeichnet wird, wird grün
```

Übungen

Experimentieren Sie!

Mit zufällig erzeugten Koordinaten lassen sich interessante Effekte erzeugen:

- 20 Linien, Anfangs- und Endpunkte zufällig
- 1000 Linien, Anfangspunkt in Bildmitte, Endpunkte zufällig
- 30 Rechtecke mit zufälligen Eckpunkten
- 50 Kreise mit zufälligen Mittelpunkten und Radien
- auch Farben lassen sich zufällig erzeugen

Zeichnen Sie konzentrische Kreise. Noch attraktiver wird das, wenn die Abstände zum Mittelpunkt hin immer enger werden.

Zeichnen Sie Diagramme, Histogramme, …

Zeichen Sie Kurven mathematischer Funktionen, z.B. y=sin(x), y=sin(x)/x, ….

Denken Sie sich eigene Beispiele aus!