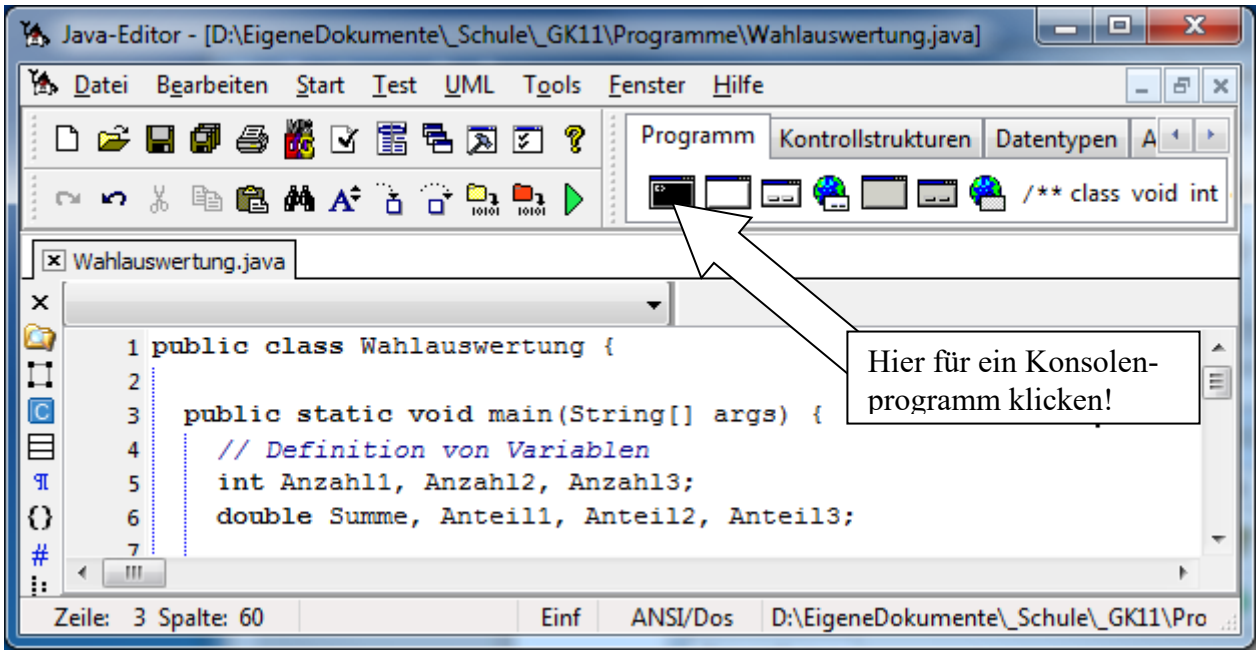
Grundlagen der Programmierung

Grundlagen der Programmierung lautet das Thema des Hessischen Lehrplans Informatik im 2. Schulhalbjahr der Einführungsphase. Wir setzen uns dabei mit grundlegenden Konzepten wie z. B. Variablen, Datentypen, Anweisungen, Kontrollstrukturen und Programmen auseinander, um die grundsätzliche Funktionsweise von Programmen zu verstehen und selbstständig Programme entwickeln zu können.

Als Programmiersprache verwenden wir Java. Zur Arbeit mit Java benötigt man das Java-Development-Kit (JDK) der Firma Sun. Als integrierte Entwicklungsumgebung (IDE, integrated development environment) verwenden wir die Hausmarke *Java-Editor*. Download und Installationshinweise findest Du unter: http://www.javaeditor.org. Zur Installation von Programmen brauchst du Administratorrechte. Nach der Grundinstallation müssen über die Konfiguration des Java-Editors noch einige Erweiterungen installiert werden: auf der Seite Compiler der Jikes-Compiler, auf der Seite Dokumentation das Manual, Tutorial und Javabuch und unter Tools die Jalopy-Erweiterung.



Klicke auf das schwarze Symbol, um ein neues Konsolen-Programm zu erzeugen. Du musst das neue Programm gleich unter einem treffenden Namen in einem Ordner speichern. Die Vorlage für das Programm sieht dann so aus:

```
public class Wahlauswertung {

  public static void main(String[] args) {

  } // end of main
} // end of class Wahlauswertung
```

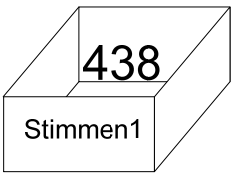
Programmiert wird zwischen den Klammern des Hauptprogramms (*main*). Mit dem gelben Symbol wird ein Programm *compiliert*, das heißt in eine für den Computer verständliche Form übersetzt. Enthält das Java-Programm Syntaxfehler (Rechtschreibfehler) wie z. B. fehlende Semikola oder falsch gesetzte Klammern, kann es nicht compiliert werden. Dann musst Du erst die Syntaxfehler beseitigen. Mit dem grünen Symbol kannst Du ein Programm starten. Der Java-Editor speichert automatisch geänderte Programme, wenn Du sie compilierst oder startest und wenn Du ein Programm startest, das noch nicht compiliert ist, so holt er das nach.

Grundbegriffe

Zur Wahl des Schulsprechers treten Max, Lea und Franz an. Mit dem folgenden Programm sollen aus den erzielten Stimmen die Stimmanteile ermittelt werden. Gib das Programm im Java-Editor ein, starte es und probiere es aus. Erläutere die Arbeitsweise des Programms.

```
public class Wahlauswertung {

  public static void main(String[] args) {
    // Definition von Variablen
    int Stimmen1, Stimmen2, Stimmen3;
    double Summe, Anteil1, Anteil2, Anteil3;

    // Eingabe
    Stimmen1 = InOut.readInt("Stimmen Max  : ");
    Stimmen2 = InOut.readInt("Stimmen Lea  : ");
    Stimmen3 = InOut.readInt("Stimmen Franz: ");

    // Verarbeitung
    Summe = Stimmen1 + Stimmen2 + Stimmen3;
    Anteil1 = Stimmen1 / Summe * 100;
    Anteil2 = Stimmen2 / Summe * 100;
    Anteil3 = 100 - Anteil1 - Anteil2;

    // Ausgabe
    System.out.println("Anteil Max  : " + Anteil1 + " %");
    System.out.println("Anteil Lea  : " + InOut.format2(Anteil2) + " %");
    System.out.println("Anteil Franz: " + InOut.formatN(Anteil3, 3) + " %");
  } // end of main
} // end of class Wahlauswertung
```

Variablen



Im Programm werden sechs Varialen definiert und dann benutzt. Eine Variable kannst Du dir als Behälter für einen Wert vorstellen. Du kannst in den Behälter einen Wert hineingeben, in wieder herausholen oder den Wert verändern. Bei der Definition einer Variablen musst Du ihren Datentyp und ihren Namen angeben. Namen beginnen mit einerm Buchstaben und müssen ohne Leerzeichen geschrieben werden.

Datentypen

Bei jeder Variablen musst du den Datentyp angeben. Die drei Variablen für die Stimmen haben den Datentyp *int*. Diesen Datentyp verwendet man, um ganzzahlige Werte zu speichern und zu verarbeiten. Die Stimmanteile lassen sich nicht mit ganzen Zahlen darstellen, hierfür braucht man Dezimalzahlen. Dafür stellt Java die Datentypen *double* zur Verfügung. Weitere wichtige Datentypen sind *String* zum Verarbeiten von Zeichenketten und *boolean* für die beiden Wahrheitswerte *true* und *false*.

Datentyp	Bedeutung	Beispiele
int	ganze Zahlen	5 -17 2345
double	Dezimalzahlen	0.765 -12.45 84.4E-07
String	Zeichenketten	“Max“ “Anteil Lea“
boolean	Wahrheitswerte	true false

EVA-Prinzip

In diesem einfachen Programm folgen drei einfache Verarbeitungsschritte aufeinander:
Eingabe – **V**erarbeitung – **A**usgabe
von Daten. Dieses Grundmuster ist typisch für Computeranwendungen. Man nennt es daher auch das *EVA-Prinzip*.

Eingabe

Die Eingabe erfolgt mit

```
Stimmen1 = InOut.readInt("Stimmen Max  : ");
```

Der String "Stimmen Max : " wird ausgegeben und mit *InOut.readInt* wird eine ganze Zahl eingegeben. Diese Zahl wird dann in der Variablen *Stimmen1* gespeichert.

Verarbeitung - Wertzuweisung

Ein Verarbeitungsschritt ist

```
Summe = Stimmen1 + Stimmen2 + Stimmen3;
```

Dabei werden die Zahlen in den Variablen Stimmen1, Stimmen2 und Stimmen3 addiert und das Ergebnis in der Variablen *Summe* gespeichert.

Ein solcher Verarbeitungsschritt kommt in Programmen oft vor. Er hat die allgemeine Form

```
Variable = Rechenausdruck;
```

und wird *Wertzuweisung* genannt, weil der links stehenden Variablen der Wert des rechts stehenden Rechenausdrucks zugewiesen wird.

Ausgabe

Die Ausgabe der Ergebnisse erfolgt über

```
System.out.println("Anteil Max  : " + Anteil1 + " %");
```

Zur Zeitersparnis kannst Du dir mit *Strg+U* den Ausgabebefehl *System.out.println();* in Dein Programm einfügen lassen. Merke Dir Strg+U! In den Klammern steht, was ausgegeben werden soll. Kommt dabei eine Variable vor, so wird nicht der Variablenname ausgegeben sondern der Wert, der in der Variablen gespeichert ist.

Bei double-Werten kannst Du mit *InOut.format2(double-Variable)* die Dezimalzahl mit zwei Nachkommastellen ausgeben lassen.

Aufgabe Maklerfirma

Eine Maklerfirma verkauft Grundstücke. Schreibe ein Programm, das für ein rechteckiges Grundstück die Längen der Seiten (in Meter) und den Quadratmeterpreis einliest. Gib dazu eine Rechnung auf dem Bildschirm aus, die die eingegebenen Daten, den Grundstückspreis, die Maklergebühr von 3%, die Mehrwertsteuer, sowie den Gesamtbetrag enthält.

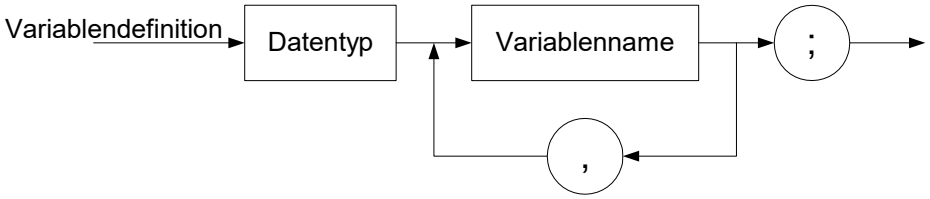
Handykosten

Karin hat zur Berechnung der monatlichen Handykosten ein Programm erstellt. Analysiere und beschreibe es. Gib das Programm ein und teste es. Vergleiche es mit dem Programm Wahlauswertung und tausche dich darüber mit anderen aus. Erweitere das Programm um einen Internet-Flat-Tarif, um einen Rabatt und um die Eingabe der Kosten pro SMS und MMS.

```
public class Handykosten {

  public static void main(String[] args) {
    // Definition von Variablen
    String Name;
    int AnzahlSMS, AnzahlMMS;
    double PreisSMS = 0.19;
    double PreisMMS = 0.39;
    double KostenSMS, KostenMMS, KostenGesamt;
    boolean Jugendlicher;

    // Eingabe
    System.out.println("--- Eingabe ---");
    Name = InOut.readString("Dein Name: ");
    AnzahlSMS = InOut.readInt("Anzahl der SMS: ");
    AnzahlMMS = InOut.readInt("Anzahl der MMS: ");
    Jugendlicher = InOut.readBoolean("Bist du Jugendlicher (j/n)? ");

    // Verarbeitung
    KostenSMS = AnzahlSMS*PreisSMS;
    KostenMMS = AnzahlMMS*PreisMMS;
    KostenGesamt = KostenSMS + KostenMMS;

    // Ausgabe
    System.out.println("");
    System.out.println("--- Ausgabe ---");
    System.out.println("Handykosten für " + Name);
    System.out.println("Kosten für " + AnzahlSMS + " SMS: " +
                       InOut.format2(KostenSMS) + " Euro");
    System.out.println("Kosten für " + AnzahlMMS + " MMS: " +
                       InOut.format2(KostenMMS) + " Euro");
    System.out.println("Gesamtkosten: " + InOut.format2(KostenGesamt) + " Euro");
    System.out.println("Du bist Jugendlicher: " + Jugendlicher);
  } // end of main
} // end of class Handykosten
```

Aufgabe Syntaxdiagramm

a) Schaue dir im Programm Wahlauswertung an, wie Variablen definiert werden. Erläutere dann das folgende Syntaxdiagramm.

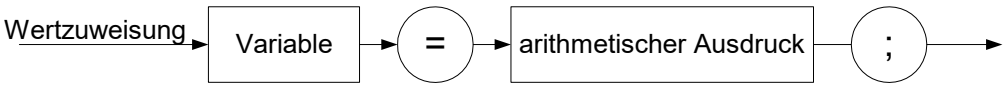


b) Eine Variable kann bei der Definition auch gleich einen Anfangswert erhalten (vergleiche Handykosten). Zeichne für diese Definition einer Variablen ein Syntaxdiagramm.

Wertzuweisung

Der Verarbeitungsteil des Programms Handykosten besteht aus drei *Wertzuweisungen*. Eine Wertzuweisung sorgt dafür, dass eine Variable einen Wert erhält bzw. dass der bisherige Wert geändert wird. Bei der Wertzuweisung `KostenGesamt = KostenSMS + KostenMMS;` wird also zunächst die Summe berechnet und dann der Wert in der Variablen *KostenGesamt* gespeichert.

Der allgemeine Aufbau einer Wertzuweisung lässt sich als Syntaxdiagramm wie folgt darstellen:



Eine Wertzuweisung fängt zwar mit der Variablen an, die einen neuen Wert erhalten soll, aber wenn der Computer eine Wertzuweisung ausführt, rechnet er erst den arithmetischen Ausdruck aus und speichert erst dann das Ergebnis in der Variablen.

Aufgabe Wertzuweisung

Welche Werte haben die Variablen *Summe* und *Produkt* bei folgender Sequenz von Wertzuweisungen?

```
int Summe, Produkt;
Summe = 2;
Summe = Summe + 1;
Produkt = Summe;
Produkt = Produkt * Produkt;
Produkt = Summe * Produkt;
Summe = Summe + Produkt;
```

Wassergebührenbescheid

Entwickle ein Programm für einen Wassergebürenbescheid. Analysiere dazu den gegebenen Gebürhenbescheid.

Eingabe: Welche Daten musst du eingeben?
Verarbeitung: Wie und in welcher Reihenfolge werden die Gebühren berechnet.
Ausgabe: Welche Daten musst du ausgeben?

Implementiere dann deine Überlegungen als Java-Programm, das einen nachvollziehbaren Wassergebührenbescheid ausgibt.

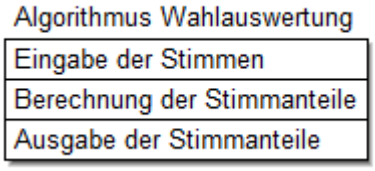
Wasser

Gebührenbescheid

Zählerstand Alt	Zählerstand Neu	Wasserverbrauch m³	monatl. Grundpreis	Preis/ pro m³	Netto Betrag (€)	Ust 7%	Brutto Betrag (€)
744	868	124		1,68	208,32	14,58	222,90
			6,90		82,80	5,80	88,60
		124			**291,12**	**20,38**	**311,50**

Struktogramm und EVA-Prinzip

Die Grundstruktur eines Programms, z. B. Wahlauswertung, lässt sich in einem sogenannten *Struktogramm* darstellen.

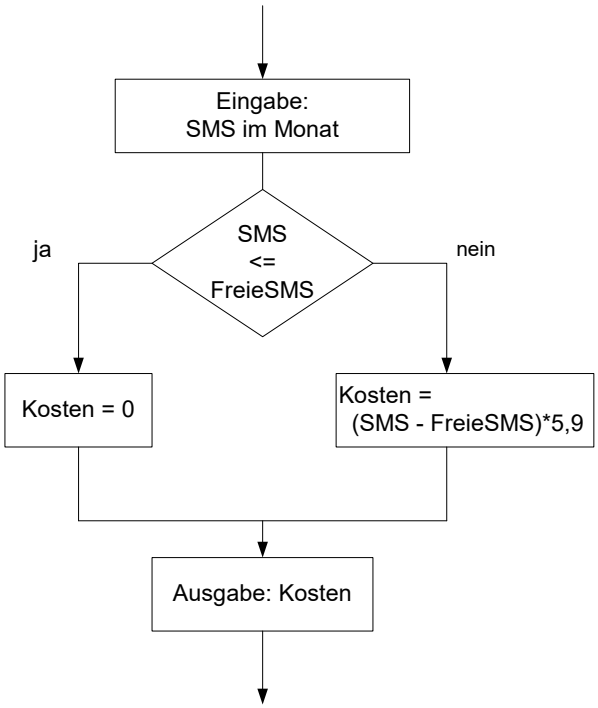


Das Programm besteht einfach aus einer Folge von Verarbeitungsschritten. Eine solche Folge nennt man *Sequenz*.

Fallunterscheidung

Beim Lösen von Anwendungsproblemen sind oft verschiedene Fälle zu unterscheiden, z. B. minder- oder volljährig, reduzierter oder voller Mehrwertsteuersatz, Anzahl verschickter SMS kleiner gleich Anzahl freier SMS, Einzel- oder Großhandelskunde, Gewicht des Fluggepäck kleiner gleich der Freigrenze 20 kg usw.

Anschaulich lässt sich eine solche Fallunterscheidung durch eine *Verzweigung* in einem Flussdiagramm darstellen. In der Raute steht das Kriterium für die Fallunterscheidung. Ist das Kriterium erfüllt verzweigt der Programmablauf nach links, anderenfalls nach rechts.

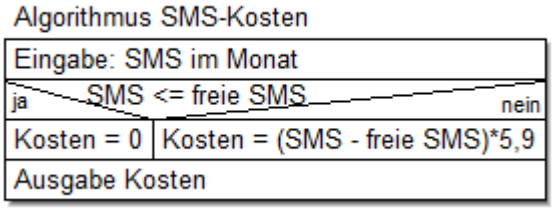


Da die Anweisungen in einem Programm mit einer Verzweigung nicht mehr wie bei einer Sequenz linear aufeinander folgen können, braucht man eine Kontrollstruktur zur Realisierung der Verzweigung. In Java ist dies die *if- Anweisung.* Das folgende einfache Beispiel zeigt, wie die Verzweigung im Flussdiagramm in Java ausgedrückt wird.

```
public class SMS {

  public static void main(String[] args) {
    // Definition von Variablen
    int AnzahlSMS;
    int FreieSMS = 25;
    double Kosten;

    // Eingabe
    AnzahlSMS = InOut.readInt("Eingabe der SMS im Monat: ");

    // Verarbeitung
    if (AnzahlSMS <= FreieSMS) {
      Kosten = 0;
    } else {
      Kosten = (AnzahlSMS - FreieSMS) * 5.9;
    } // end of if-else

    // Ausgabe
    System.out.println("Kosten: " + Kosten + " Cent");
  } // end of main
} // end of class SMS
```

Struktogramm für die Fallunterscheidung

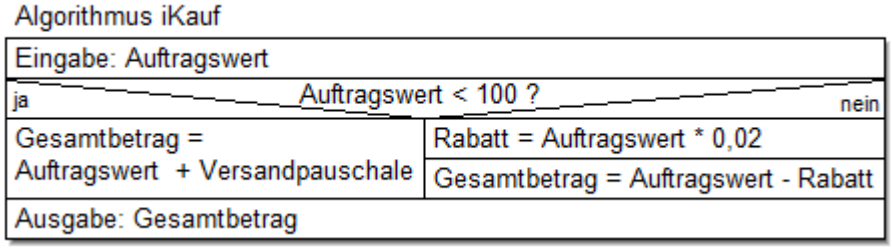
Fallunterscheidungen können statt mit einem Flussdiagramm auch mit einem Struktogramm dargestellt werden. Für obiges Beispiel sieht das dann wie im Bild aus.



Aufgabe iKauf

Beim Online-Kaufhaus iKauf zahlt man bei einem Auftragswert unter 100 € die Versandpauschale 3,50 €. Ab 100 € entfällt die Pauschale, zudem erhält man dann einen Rabatt von 2 %. Nach Eingabe des Auftragswertes soll der Gesamtbetrag ermittelt werden.

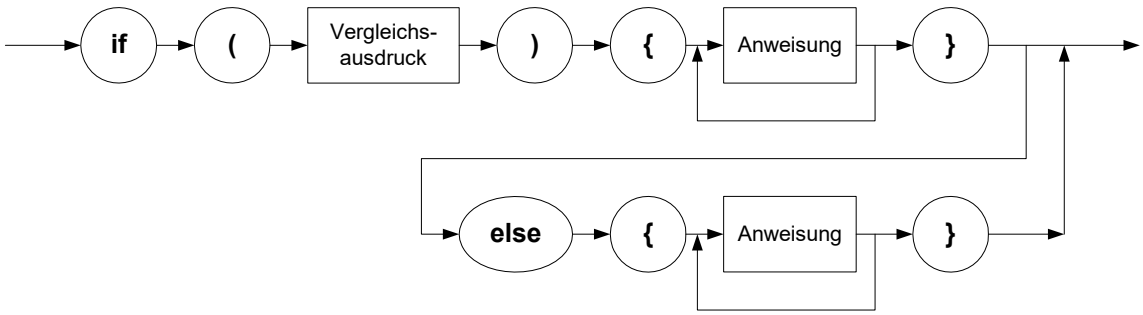
a) Analysiere das Struktogramm zur Problemlösung.
b) Konstruiere das Struktogramm mit dem Struktogramm-Editor (siehe Moodle).
c) Implementiere ein Programm gemäß diesem Algorithmus.



Syntaxdiagramm

Der Java-Editor unterstützt dich beim Schreiben einer if-Anweisung. Platziere dazu in deinem Java-Programm den Cursor an die Stelle, wo die if-Anweisung stehen soll. Öffne dann die Registerkarte Kontrollstrukturen und klicke auf „if..else“. Dadurch wird die Struktur einer if-Anweisung im Java-Programm erzeugt, welche du dann ausfüllen kannst.

Das folgende Syntaxdiagramm zeigt dir, wie eine if-Anweisung syntaktisch aufgebaut ist.

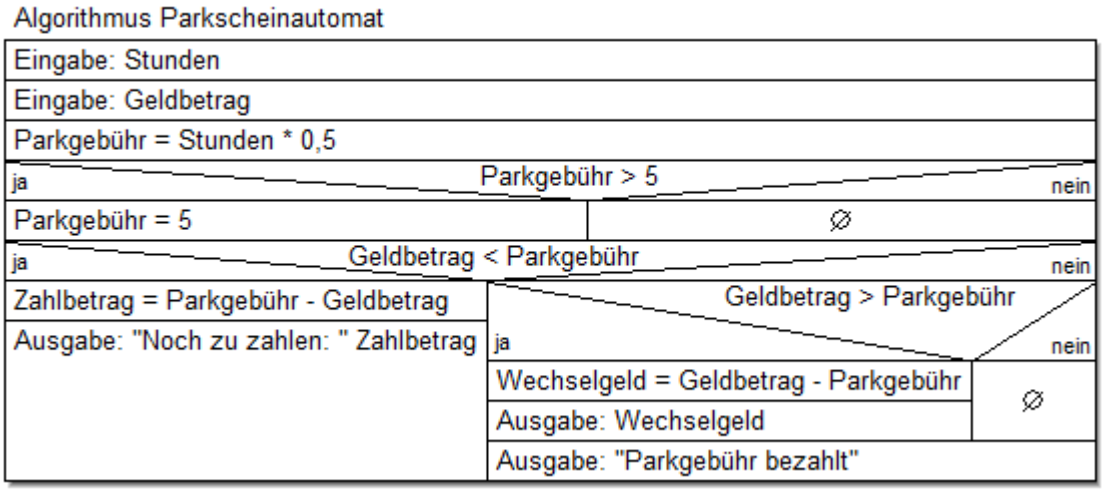


Aufgabe Syntaxdiagramm if-Anweisung

a) Erläutere dieses Syntaxdiagramm allgemein und am Programmbeispiel SMS.
b) Vergleiche das Syntaxdiagramm mit der Struktur, die der Java-Editor beim Anklicken auf „if..else“ erzeugt.

Aufgabe Parkscheinautomat

Auf einem kostenpflichtigen Parkplatz gilt der Tarif 0,5 € pro angefangener Stunde, maximal 5 € pro Tag. Bezahlt wird beim Verlassen des Parkplatzes. Die Funktionsweise des Parkscheinautomaten wird durch nachfolgendes Struktogramm beschrieben. Es enthält drei Verzweigungen. Bei der ersten handelt es sich um eine einseitige Fallunterscheidung. Die beiden anderen sind ineinander verschachtelt. Dadurch kann man mehr als zwei Fälle unterscheiden, in diesem Beispiel die drei Fälle Geldbetrag <, = und > Parkgebühr.



Gemäß dem Struktogramm kann der Parkscheinautomat durch folgendes Programm simuliert werden.

```
public class Parkscheinautomat {

  public static void main(String[] args) {
    int Stunden;
    double Geldbetrag, Parkgebühr, Zahlbetrag, Wechselgeld;

    Stunden = InOut.readInt("Parkzeit in Stunden: ");
    Geldbetrag = InOut.readDouble("Eingeworfener Geldbetrag in Euro: ");

    Parkgebühr = Stunden * 0.5;
    if (Parkgebühr > 5) {
      Parkgebühr = 5;
    } // end of if
    ...
  } // end of main
} // end of class Parkscheinautomat
```

a) Vergleiche Struktogramm und Programm.
b) Vervollständige das Programm.

Aufgabe Body-Mass-Index

Informiere dich in der Wikipedia über den Body-Mass-Index.

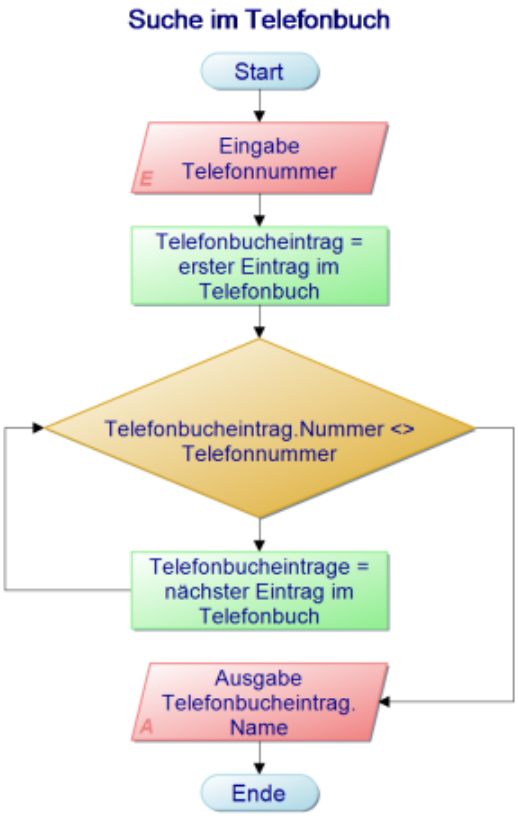
a) Entwickle ein Struktogramm zur Ermittlung der Gewichtskategorien: Untergewicht, Normalgewicht und Übergewicht. Erstelle dazu ein Java-Programm.
b) Optional: Das Programm soll das altersabhängige Normalgewicht berücksichtigen.

Aufgabe Bahnreise

Die Bahn bietet Gruppenreisen ab 6 Personen im Tarif Gruppe&Spar 70 an. Dabei gibt es auf den Normalpreis für eine Person 70 % Rabatt, maximal bezahlt man aber 36,60 € pro Person, egal wie weit man innerhalb Deutschlands reist. Entwickle einen Preisrechner für solche Gruppenreisen. Eingegeben werden der Normalpreis für eine Person und die Anzahl der in der Gruppe reisenden Personen, ausgegeben wird der Preis für die Gruppe.

Wiederholungen

Die Mächtigkeit des Computers beruht letztlich darauf, dass er Vorgänge millionenfach exakt und in atemberaubender Geschwindigkeit wiederholen kann. Als Mensch kann man sich noch sehr anstrengen, irgendwann lässt die Konzentration nach und es schleichen sich Fehler ein. Niemand schafft es z. B. zu einer Telefonnummer aus dem Darmstädter Telefonbuch den zugehörigen Namen herauszusuchen. Für einen Computer ist das kein Problem. Er liest der Reihe nach einen Telefonbucheintrag nach dem anderen bis er den richtigen gefunden hat und gibt dann den zugehörigen Namen aus. Dies ist im nebenstehenden Flussdiagramm grafisch dargestellt.



Verfolgt man die möglichen Wege im Flussdiagramm, so wird deutlich, dass es im linken Teil einen Rundkurs, eine sogenannte *Schleife*, gibt. Diese Schleife wird solange durchlaufen, bis der richtige Telefonbucheintrag gefunden ist.

Zum Programmieren einer Schleife braucht man eine Kontrollstruktur, die automatisch die gewünschten Wiederholungen im Flussdiagramm durchführt. In Java ist dies die *while-Anweisung*. Das folgende einfache Beispiel eines Ratespiels zeigt, wie die Schleife in Java ausgedrückt wird.

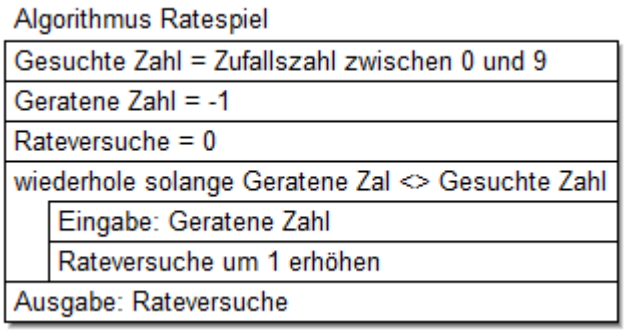
```
public class Ratespiel {

  public static void main(String[] args) {
    int GesuchteZahl = (int) (Math.random()*10);
    int GerateneZahl = -1;
    int Rateversuche = 0;
    while (GerateneZahl != GesuchteZahl) {
      GerateneZahl = InOut.readInt("Gib eine Zahl ein: ");
      Rateversuche = Rateversuche + 1;
    }
    System.out.print("Du hast die gesuchte Zahl " + GesuchteZahl);
    System.out.println(" nach " + Rateversuche + " Versuchen geraten. ");
  } // end of main
} // end of class Ratespiel
```

Zunächst wird mit Hilfe eines Zufallsgenerators eine zufällige Zahl zwischen 0 und 9 bestimmt, die es zu erraten gilt. Die geratene Zahl wird auf -1 gesetzt und die Anzahl der Rateversuche mit 0 initialisiert. Solange (engl. while) sich die geratene Zahl ungleich der gesuchten Zahl ist, muss der Spieler eine Zahl eingeben. Jeder Rateversuch wird gezählt. Wenn die geratene Zahl mit der gesuchten Zahl übereinstimmt, wird die Schleife verlassen und es die Anzahl der Rateversuche ausgegeben.

Aufgabe
In dieser Form ist das Ratespiel etwas langweilig. Besser wird es, wenn man in der Schleife ausgibt, ob die geratene Zahl kleiner oder größer als die gesuchte Zahl ist. Ergänze eine entsprechende Fallunterscheidung. Durch intelligentes Fragen kann man jetzt viel schneller die Zahl erraten, auch wenn du den Zahlbereich von 10 auf 100 vergrößerst.

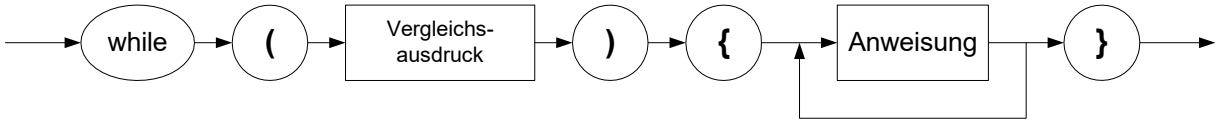
Struktogramm für Wiederholungen

Wiederholungen können statt mit einem Flussdiagramm auch mit einem Struktogramm dargestellt werden. Für obiges Beispiel sieht das dann wie im Bild aus.



Syntaxdiagramm

Auf der Registerkarte *Kontrollstrukturen* des Java-Editors findest Du natürlich auch die *while*-Anweisung. Beim Anklicken werden das Schlüsselwort *while* und die erforderlichen Klammern in den Quelltext übernommen. Den Rest musst du ausfüllen. Das Syntaxdiagramm zeigt dir den erforderlichen Aufbau.



Aufgabe 1
Ermittle mit Papier und Bleistift was das folgende Programm ausgibt.

```
public static void main(String[] args) {
  int y;
  int x = 1;
  int total = 0;
  while (x <= 5) {
    y = x * x;
    System.out.println(y);
    total = total + y;
    x = x + 1;
  }
  System.out.println("Total ist: " + total);
}
```

Aufgabe 2
Die folgende Schleife soll die Ausgabe 121 81 49 25 9 1 erzeugen. Ergänze den fehlenden Programmcode.

```
public static void main(String[] args) {
  int Zahl = 11;
  while (  ?   ) {
    System.out.print( ? + " ");
    Zahl = ? ;
  }
}
```

Simulation eines Fahrscheinautomaten

Ein Fahrscheinautomat gibt Wechselgeld in Münzen zurück und soll dabei möglichst wenige Münzen benutzen. Bei einem Fahrpreis von 13,25 € und einer Eingabe von 20 € gibt er als Wechselgeld drei 2 €-Münzen, eine 50 Cent, eine 20 Cent und eine 5 Cent-Münze zurück. Mit folgendem Programm kann der Fahrscheinautomat simuliert werden.
a) Analysiere die Funktionsweise des Programms.
b) Vervollständige das Programm.
c) Stelle das Programm in Struktogrammform dar.

```
public class Fahrscheinautomat {

  public static void main(String[] args) {
    double Fahrpreis, Gezahlt, Wechselgeld;
    Fahrpreis = InOut.readDouble("Fahrpreis: ");
    Gezahlt   = InOut.readDouble("Gezahlt  : ");
    while (Gezahlt < Fahrpreis) {
      double ZuZahlen = Fahrpreis - Gezahlt;
      System.out.println("Noch " + ZuZahlen + " zu zahlen!");
      Gezahlt = Gezahlt + InOut.readDouble("Zahle: ");
    } // end of while
    Wechselgeld = Gezahlt - Fahrpreis;
    if (Wechselgeld > 0) {
      System.out.println("Wechselgeld" + Wechselgeld);
      System.out.print("Rückgabe: ");
      while (Wechselgeld >= 2) {
        System.out.print("2 Euro ");
        Wechselgeld = Wechselgeld - 2;
      } // end of while
      if (Wechselgeld >= 1) {
        System.out.print("1 Euro ");
        Wechselgeld = Wechselgeld - 1;
      } // end of if
      // hier ergänzen
    } else {
      System.out.println("Passend bezahlt");
    } // end of if-else
  } // end of main
} // end of class Fahrscheinautomat
```

Verflixte Dezimalzahlen

Beträgt der Fahrpreis 4.40 € und wirft man 10 € ein, so sollte man 5.60 € als Wechselgeld erhalten, der Fahrscheinautomat gibt aber nur 5.59 € zurück. Im Einzelschrittmodus findet man heraus, dass bei der ersten Subtraktion 5.60 – 2 = 3.5999999999999996 berechnet wird und nicht 3.6. Das führt schließlich dazu, dass der letzte Cent nicht ausgegeben wird.

1	1	0	0
2	0,5	1	0,5
4	0,25	0	0
8	0,125	0	0
16	0,0625	1	0,0625
32	0,03125	1	0,03125
64	0,015625	0	0
128	0,0078125	0	0
256	0,00390625	1	0,00390625
512	0,00195313	1	0,00195313
1024	0,00097656	0	0
	Summe		0,59960938

Computer stellen Zahlen im Zweiersystem dar, weil eine Speicherzelle geladen oder ungeladen sein kann. Im Zweiersystem hat 3.6 die periodische Darstellung 11.100110011001… wobei beim Datentyp double nur 64 Binärstellen bzw. 16 Dezimalstellen gespeichert werden. Die im Zehnersystem exakt darstellbare Zahl 3.6 wird vom Computer also nur näherungsweise dargestellt.

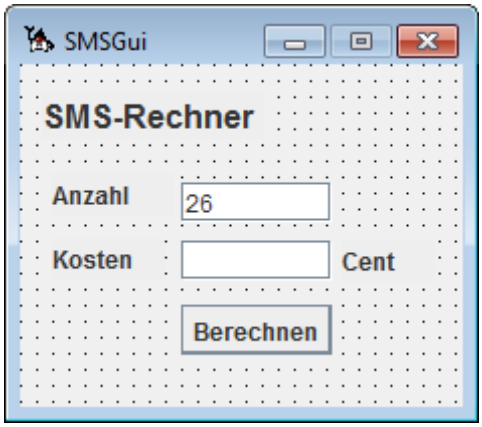
Wir erhöhen daher das berechnete Wechselgeld um einen Tausendstel Cent, damit es richtig ausgegeben wird.

Simulation einer Population
In einem Teich leben anfangs A = 50 Karpfen. Platz ist für höchstens H = 350 vorhanden. Die Karpfen vermehren sich jährlich mit einer Rate von 6% der Differenz zwischen Höchstbestand H und aktuellem Bestand A. Wie entwickelt sich die Population
1. ohne äußere Einwirkungen
2. wenn am Ende jedes Jahres 20% des Bestandes entnomen werden?

a) Berechne den Karpfenbestand für das erste und zweite Jahr mit dem Taschenrechner.
b) Erstelle ein Struktogramm, zur wiederholten Berechnung des Bestandes.
c) Schreibe ein Simulationsprogramm, mit dem man den Verlauf der Bestandsentwicklung verfolgen kann.

GUI-Programmierung

Konsolenprogramme spielen bei der automatischen Datenverarbeitung eine große Rolle. Für die interaktive Arbeit mit Computern benutzt man eher GUI (Graphical User Interface) Programme.

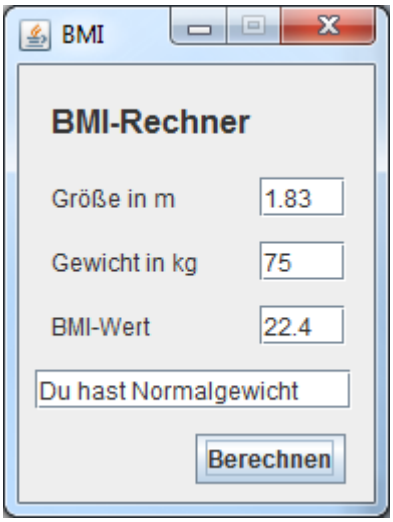


Für das SMS-Programm von Seite 6 kann ein GUI-Formular wie gezeigt gestaltet werden. Es besteht aus vier Label-Komponenten, zwei NumberField-Komponenten und einer Button-Komponente. Beim Anklicken des Buttons *Berechnen* werden die Anzahl der SMS eingelesen (Eingabe), die Kosten berechnet (Verarbeitung) und die Kosten in der unteren NumberField-Komponente ausgegeben (Ausgabe).

```
// Ereignisprozedur des Buttons bBerechnen
public void bBerechnen_ActionPerformed(ActionEvent evt) {
  // Definition von Variablen
  int AnzahlSMS;
  int FreieSMS = 25;
  double Kosten;

  // Eingabe über das NumberField nfAnzahl
  AnzahlSMS = nfAnzahl.getInt();

  // Verarbeitung
  if (AnzahlSMS <= FreieSMS) {
    Kosten = 0;
  } else {
    Kosten = (AnzahlSMS - FreieSMS) * 5.9;
  } // end of if-else

  // Ausgabe über das NumberField nfKosten
  nfKosten.setDouble(Kosten, 2);
} // end of button1_ActionPerformed
```

BMI-Rechner als GUI-Programm

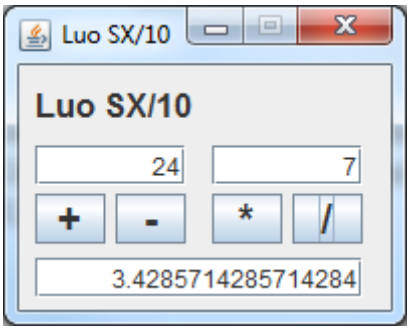
```
public void bBerechnen_ActionPerformed(ActionEvent evt) {
  // Eingabe
  String Ausgabe;
  double Groesse = nfGroesse.getDouble();
  double Gewicht = nfGewicht.getDouble();

  // Verarbeitung
  double BMI = Gewicht/(Groesse*Groesse);
  if (BMI < 18.5) {
    Ausgabe = "Untergewicht";
  } else {
    if (BMI > 25) {
      Ausgabe = "Übergewicht";
    } else {
      Ausgabe = "Normalgewicht";
    } // end of if-else
  } // end of if-else

  // Ausgabe
  nfBMI.setDouble(BMI, 1);
  tfAusgabe.setText("Du hast " + Ausgabe);
} // end of bBerechnen
```



Aufgaben
Erstelle für den SMS- und BMI-Rechner GUI-Programme.

Taschenrechner

Vor 39 Jahren gabe es in Deutschland den ersten Taschenrechner zu kaufen. Er kostete 300 DM und beherrschte lediglich die vier Grundrechenarten. Wir erstellen selbst einen solchen Taschrechner, das Modell Luo SX/10, als GUI-Programm.



Das Erstellen eines GUI-Programms ist tükisch und man kann dabei schnell einen Fehler machen. Es gibt aber Strategien zur Vermeidung von Fehlern.

1. Plane den Aufbau der GUI-Oberfläche und beachte dabei das EVA-Prinzip.

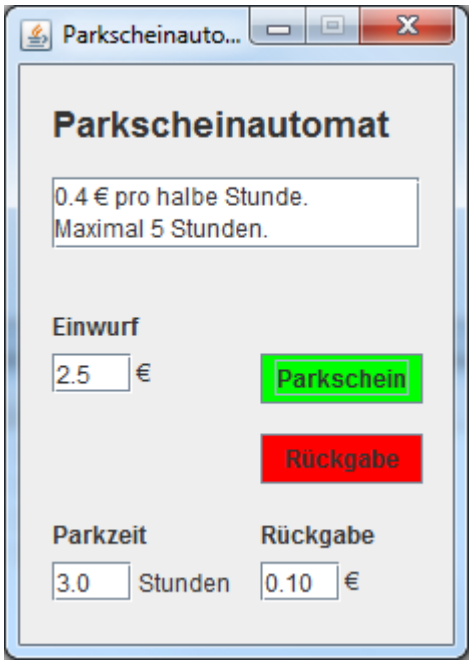
2. Benenne deine GUI-Komponenten systematisch. Alle NumberField-Komponenten erhalten einen Namen, der mit *nf* für Numberfield beginnt. Für den Taschenrechner also nfLinks, nfRechts und nfErgebnis. Alle Button-Komponenten erhalten einen Namen, der mit *b* beginnt, also bPlus, bMinus, bMal und bDurch.

3. Im Java-Quelltext wird nur dort programmiert, wo es heißt
   `// TODO hier Quelltext einfügen`
   Durch Doppelklick auf einen Button kommt man an die entsprechende Stelle.

4. Beim Programmieren geht man nach dem EVA-Prinzip vor. Also erst die Eingabe aus den NumberField-Komponenten, dann die interne Verarbeitung und zum Schluss Ausgabe der Ergebnisse.

Parkscheinautomat

a) Auf einem öffentlichen Parkplatz steht der abgebildete Parkscheinautomat. Simuliere ihn durch ein GUI-Programm mit dem abgebildeten Formular.



b) Vergleiche Konsolen- und GUI-Programme. Welche grundsätzlichen Unterschiede kannst du feststellen?

Verflixte ganze Zahlen

Probleme beim Umgang mit Dezimalzahlen haben wir schon kennengelernt. Aber auch bei Berechnungen mit ganzen Zahlen ist man vor Überraschungen nicht gefeit! Betrachten wir dazu diese Berechnung:

```
int halbeStunden = (int) (Einwurf / 0.4);
double Parkzeit = halbeStunden / 2;
```

Wirft man nun drei Euro in den Fahrscheinautomaten ein, so müssten wegen 3 / 0.4 = 7.5, (int) 7.5 = 7 und 7 / 2 = 3.5 Stunden Parkzeit berechnet werden. Das Programm gibt aber nur 3 Stunden aus.

```
int halbeStunden = (int) (3/0.4);
double Parkzeit = halbeStunden / 2;

7/2
3
7.0/2
3.5
```

Name	Typ	Wert
halbeStunden	int	7
Parkzeit	double	3.0

Wenn halbeStunden den Datentyp *int* hat, dann wird `halbeStunden / 2` als Ganzzahlberechnung durchgeführt, weil sowohl der Dividend als auch der Divisor ganzzahig ist, das Ergebnis ist dann auch ganzzahlig. Das Ergebnis 7 durch 2 ist also 3 und wird dann der double-Variablen Parkzeit zugewiesen. Hat halbeStunden den Datentyp double wird hingegeben eine Rechnung mit Dezimalzahlen durchgeführt mit dem richtigen Ergebnis 3.5. Man kann die Ganzzahldivision vermeiden, wenn man den Divisor 2 zu einer Dezimalzahl macht, also 2.0 schreibt.

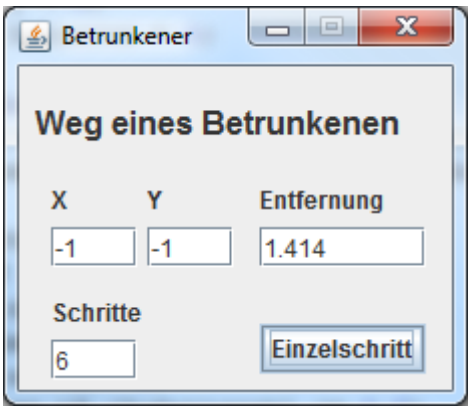
Simulationen

Simulationen sind ein wichtiges Anwendungsgebiet der Informatik. Neue Produkte werden am Computer entworfen und ihre Eigenschaften im Modell durch Simulationen überprüft, bevor erste Prototypen hergestellt werden. Sind keine analytischen Methoden für die Simulation verfügbar, so kann man Zufallszahlen verwenden (Monte-Carlo-Methode).

Weg eines Betrunkenen

Ein Betrunkener verlässt morgens früh den Club an der Positon (0/0) und wankt bei jedem Schritt zufällig in eine der vier Himmelsrichtungen. Wie weit ist er nach n Schritten vom Club entfernt?

Die aktuelle Position des Betrunkenen halten wir in den Variablen x und y fest. Zur Ausführung eines Einzelschritts bestimmen wir mittels *Math.random()* eine Zufallszahl zwischen 0 und 1. Ist diese kleiner gleich 0.25 wird die x-Position um 1 vergrößert, liegt sie zwischen 0.25 und 0.5 wird die x-Position um 1 verkleinert. Für Werte größer 0.5 wird entsprechend die y-Position verändert.



Damit die Werte für x, y und Schritte zwischen zwei Einzelschritten erhalten bleiben, müssen diese außerhalb der Ereignisprozedur definiert werden.

```
int x = 0;
int y = 0;
int Schritte = 0;

// Anfang Methoden
public void bEinzelschritt_ActionPerformed(ActionEvent evt) {
  double Zufallszahl = Math.random();

  Schritte++;
  if (Zufallszahl <= 0.25)
    x = x + 1;
  else if (Zufallszahl <= 0.5)
    x = x - 1;
 ...
```

Die Entfernung kann über den Phythagoras und Math.sqrt(…) (square root = Quadratwurzel) berechnet werden.

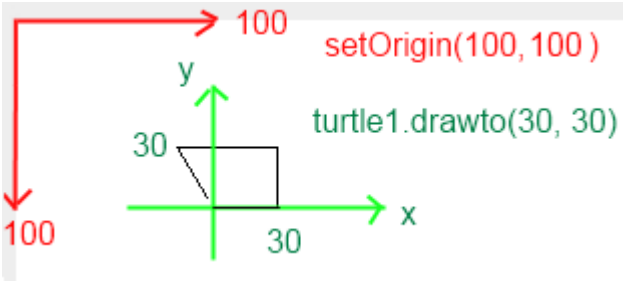
Aufgabe
Erstelle das GUI-Formular und vervollständige die Ereignisprozedur für den Einzelschritt.

Turtle

Wenn ein Computer zur Simulation von Vorgängen eingesetzt wird, so werden die Simulationsergebnisse oft visualisiert, um sie anschaulich interpretieren zu können. Für die grafische Darstellung des Weges verwenden wir die Turtle-Komponente (engl. Turtle = Schildkröte).

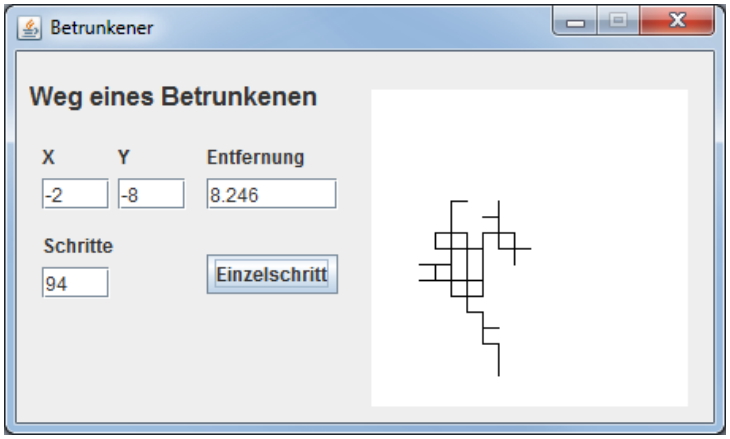
Die Position der Turtle wird durch ihre x- und y-Koordinate beschrieben. Zudem wird ihre aktuelle Bewegungsrichtung durch einen Winkel w beschrieben, wobei 0° der Richtung nach rechts entspricht. Man kann die Turtle bewegen lassen, wobei sie ihre Spur auf der Zeichenfläche hinterlässt. Das GUI-Formular stellt die Zeichenfläche dar. Anfangs befindet sich die Turtle in der Mitte ihrer Zeichenfläche. Mit den folgenden Methoden kann die Turtle programmiert werden.

turn (double angle)	Ändert die Zeichenrichtung relativ um den Winkel angle.
turnto (double angle)	Legt die Zeichenrichtung absolut fest
draw (double length)	Die Turtle zeichnet in der aktuellen Richtung eine Strecke der Länge *length*.
drawto(double x, double y)	Die Turtle zeichnet von der aktuellen Position eine Strecke zum Punkt P(x/y) im **grünen** Koordinatensystem
move (double length)	Bewegt die Turtle in der aktuellen Zeichenrichtung um eine Strecke der Länge *length* ohne zu zeichnen.
moveto (double x, double y)	Setzt die Turtle auf den Punkt P(x/y)
setForeground(Color c)	Setzt die Farbe c als Zeichenfarbe
setBackground(Color c)	Setzt die Farbe c als Hintergrundfarbe
clear()	Löscht die Zeichenfläche.
setOrigin(double x, double y)	Legt den Ursprung des **grünen** Koordinatensystems der Turtle auf der Zeichenfläche fest. Die Zeichenfläche hat ihren eigenen Ursprung immer links oben im **roten** Koordinatensystem.




```
turtle1.setOrigin(100, 100);
turtle1.drawto(30, 0);
turtle1.drawto(30, 30);
turtle1.drawto(-20, 30);
turtle1.drawto(-5, 5);
```

Jeden Einzelschritt kann man mit turtle1.drawto(10*x, 10*y); zeichnen lassen.



Viele Einzelschritte

Es ist mühsam in Einzelschritten vorzugehen. Einfacher ist es, wenn wir die gewünschte Anzahl von Schritten eingeben und dann die Ereignisprozedur diese Schritte ausführen lassen. Das geht mit einer **while**-Schleife, passender ist aber eine **for**-Schleife. for-Schleifen setzt man ein, wenn man genau weiß, wie oft die Schleife durchlaufen werden muss.

Die Syntax ist etwas kompliziert, die Anwendung aber einfacher als bei der while-Schleife.

```
for (Initialisierung; Schleifenbedingung; Veränderung der Schleifenvariablen) {
  Anweisungen
}
```

Im Initialisierungsteil wird die Schleifenvariable initialisiert, bei erfüllter Schleifenbedingung wird die Schleife nochmals ausgeführt und zum Schluss wird die Schleifenvariable hoch- oder runter gezählt.

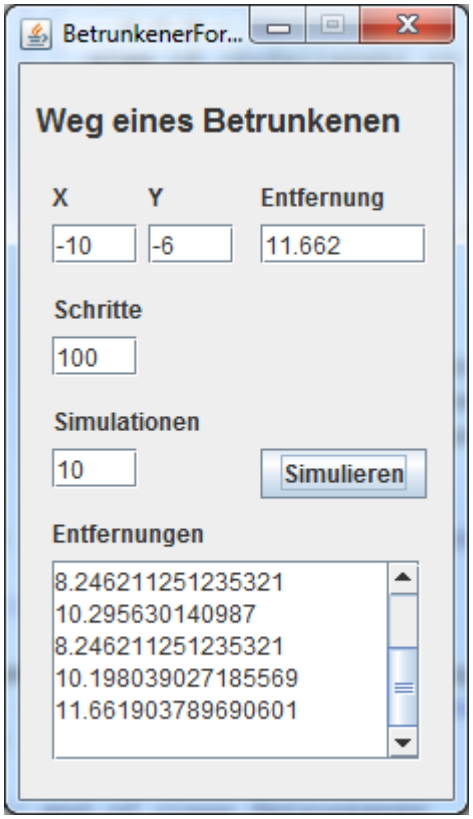
Beispiele

```
for (int i = 0; i < 10; i++) { System.out.println(i); }

int k;
for (k = 10; 0 < k; k = k - 2) { System.out.println(k); }
System.out.println(k);
```

Aufgabe
Verändere das Programm nun so, dass die Anzahl der eingegebenen Schritte ausgeführt wird. Überlege dir gut, was in die Schleife gehört und was außerhalb stehen muss.

```
public void bEinzelschritt_ActionPerformed(ActionEvent evt) {

  Schritte = nfSchritte.getInt();
  for (int i = 1; i <= Schritte; i++) {
    // bisheriges Programm entsprechend angepasst
  }
```

Mehrere Simulationen
Wir können nun zwar problemlos 100 oder 1000 Schritte ausführen lassen, aber was können wir aus einer Simulation erkennen? Recht wenig, man muss schon mehrmals simulieren, um die Ergebnisse interpretieren zu können.



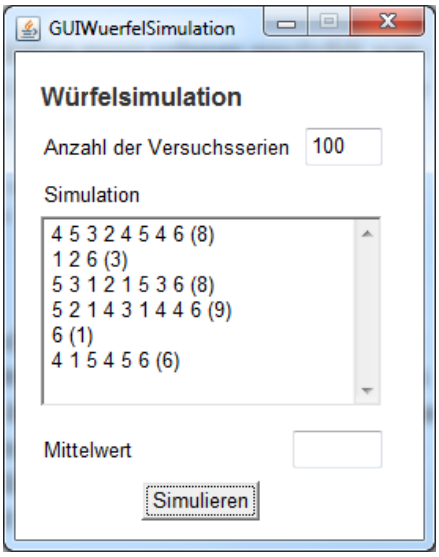
In einem nächsten Entwicklungsschritt erweitern wir das Programm so, dass es eine gewünschte Anzahl von Simulationen ausführen kann. Eine 100-Schritt-Simulation wird im Beispiel zehnmal ausgeführt. Dies erreicht man durch eine weitere for-Schleife um die bisherige for-Schleife.

Zur Anzeige der Simulationsergebnisse setzen wir eine TextArea-Komponente ein, welche mehrere Textzeilen anzeigen kann. Im Beispiel sieht man die Ergebnisse der fünf letzten Simulationen. Wenn die TextArea-Komponente den Namen taAusgabe hat, so wird eine Zeile ausgegeben mit:

```
taAusgabe.append(Entfernung + "\n");
```

Würfelsimulation

Es ist ein Programm zu entwickeln, welches das Werfen eines Würfels simuliert. Wir interessieren uns dabei für die Frage, wie oft man im Mittel würfeln muss, bis eine 6 erscheint. Das Endergebnis könnte wie im Bild dargestellt aussehen.



Für eine Simulation muss solange gewürfelt werden, bis eine 6 kommt. Dies macht man mit einer while-Schleife:

```
while (Wurf !=6) {
  Anweisungen
}
```

Aufgabe Würfelsimulation

a) Analysiere und implementiere diese Prozedur:

```
public void bSimulieren_ActionPerformed(ActionEvent evt) {
  int Wurf, WuerfeBis6;
  String Wurfserie;
  Wurfserie = "";
  WuerfeBis6 = 0;
  Wurf = 0;
  while (Wurf != 6) {
    Wurf = (int) (Math.random()*6) + 1;
    WuerfeBis6++;
    Wurfserie = Wurfserie + " " + Wurf;
  } // end of while
  taAusgabe.append(Wurfserie + " (" + WuerfeBis6 + ")\n");
}
```

Es sollen nun so viele Versuche durchgeführt werden, wie der Benutzer eingibt. Also brauchen wir noch eine äußere Schleife, die dafür sorgt, dass die gewünschte Anzahl von Versuchsdurchführungen ausgeführt und ausgewertet werden. Dies macht man mit einer for-Schleife:

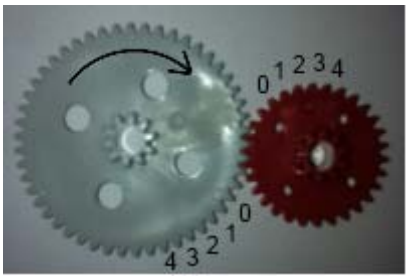
```
for (int i = 1; i <= AnzahlVersuchsserien; i++) {
  Anweisungen
}
```

b) Programmiere dies gemäß diesem Struktogramm:

Algorithmus Würfelsimulation

Eingabe: Anzahl der Simulationen
Summe = 0
wiederhole für i=1 bis Anzahl der Simulationen
Wurf = 0
wiederhole solange Wurf <> 6
Wurf = Zufallszahl zwischen 1 und 6
WuerfeBis6 = WuerfeBis6 + 1
Wurfserie = Wurfserie + " " + Wurf
Ausgabe: Wurfserie
Summe = Summe + WuerfeBis6
Mittelwert berechnen und ausgeben

Eine zündende Idee!

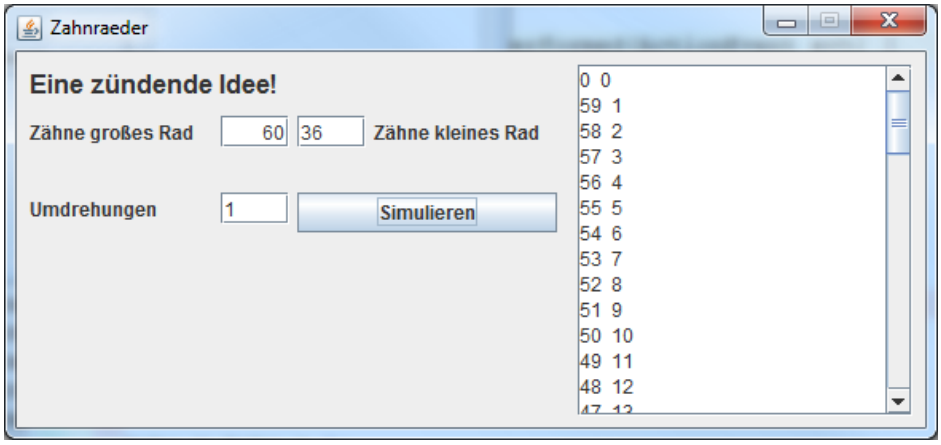


Zum Jahreswechsel 220/219 v. Chr. soll Archimedes auf Weisung der Stadtväter von Syrakus das große Feuerwerk organisieren. Hierzu konstruiert er eine Vorrichtung aus zwei Zahnrädern, die das Feuerwerk pünktlich zünden soll. Das größere Rad dreht sich einmal pro Minute. Es hat 60 Zähne und treibt ein kleineres mit 36 Zähnen an. Die Zähne des kleinen Rades und die Lücken des großen numeriert er mit 0 beginnend im Uhrzeigersinn.

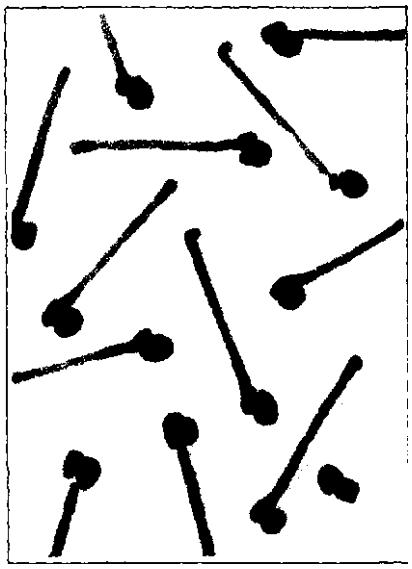
Über je einen Feuerstein, der bei dem großen Rad in Lücke 0 sitzt und bei dem kleinen an einem beliebigen Zahn befestigt werden kann, ist es möglich, eine Zündung auszulösen. Diese erfolgt, wenn die beiden Feuersteine aufeinander schlagen. Bei jedem Zündversuch stellt er zu Beginn Zahn 0 des kleinen Rades auf Lücke 0 des großen Rades. Er überlegt sich, dass nach 5 Umdrehungen des großen Rades Zahn 12 des kleinen Rades auf Lücke 0 des großen Rades trifft, denn 5*60=300 und 300 = 36*8 +12. Also befestigt er den Feuerstein auf Zahn 12, doch statt nach 5 Minuten zündet es bereits nach 2. Doch kaum, dass er sich an seinen versengten Bart fasst, ruft er "Heureka, ich hab's!" und macht sich daran, ein kleines Rad mit 37 Zähnen anzufertigen. (Quelle: Bundeswettbewerb Informatik)

Aufgabe:
Schreibe ein Programm, das folgendes leistet:

Eingegeben werden die Anzahlen der Zähne des großen und kleinen Rads sowie eine weitere Größe.
Nach Eingabe der Umdrehungszahl des großen Rads simuliert das Programm die Drehbewegung, indem es alle Paare Lücke, Zahn ausgibt, die sich im Ablauf treffen. Gib bei einer Zündung den Zündzeitpunkt aus.



Streichholzspiel



Am Streichholzspiel nehmen zwei Spieler teil. Zu Beginn liegen dreizehn Streichhölzer auf dem Tisch. Die Spieler nehmen abwechselnd Streichhölzer weg, mindestens eins und höchstens drei. Wer das letzte Streichholz nehmen muss, der hat verloren.

Das Streichholzspiel kann aber auch mit beliebig vielen Streichhölzern gespielt werden. Die Mitspieler einigen sich dann vor dem Spiel, mit wie vielen Streichhölzern sie spielen wollen und wie viele Streichhölzer in einem Zug höchstens weggenommen werden dürfen.

a) Schreibe ein Programm mit dem zwei Spieler am Computer gegeneinander spielen können. Eingabe für das Programm sind die Gesamtzahl der Streichhölzer zu Beginn und die Höchstzahl der Streichhölzer, die pro Zug weggenommen werden dürfen. Der Computer achtet darauf, dass nicht geschummelt wird. Er gibt an, wer jeweils dran ist und ermittelt den Gewinner.

b) Ändere das Programm so, dass Du das Streichholzspiel gegen Deinen Computer spielen kannst. Das Programm soll immer versuchen, zu gewinnen.

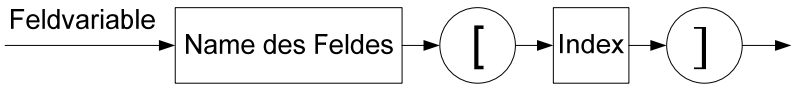
(Quelle: Bundeswettbewerb Informatik)

Felder

In Computeranwendungen hat man es häufig mit vielen gleichartigen Daten zu tun, z. B. Sammlung von Telefonnummern, E-Mail-Adressen, Favoriten, Rechnungsdaten, Messwerte, Buchungen oder auch Aktienkurse. Zum Speichern und Verarbeiten gleichartiger Daten benutzt man ein *Feld*.

Ein Feld (engl. array) ist eine Reihe von Variablen gleichen Datentyps. Die einzelnen Variablen können über einen Index angesprochen werden, der in rechteckigen Klammern geschrieben wird. In Feldern kann man eine begrenzte Anzahl gleichartiger Daten speichern, also zum Beispiel nur int-Zahlen oder nur Strings. Im Gegensatz zu *einfachen* Datentypen wie int, double, char oder boolean hat ein Feld einen *strukturierten* Datentyp. Die Struktur eines Feldes wird im Bild veranschaulicht.

Das Feld hat den Namen *Zahl* und besteht aus vier Variablen des Datentyps *int*. Alle Variablen haben die gleiche Größe. Der Name *einer* Variablen des Feldes wird gemäß folgendem Syntaxdiagramm gebildet:



Die erste Variable hat den Namen *Zahl[0]* und 0 als Index, die vierte den Namen *Zahl[3]* und 3 als Index. Die Variablennamen beginnen also alle mit dem Namen des Feldes und unterscheiden sich im Index. Der Index beginnt immer bei 0. Die Variable Zahl[0] hat den Wert 17. Die Variable Zahl[1] hatte anfangs den Wert 29, wurde durch eine Wertzuweisung auf 8 und dann auf 13 geändert.

Beim Erzeugen eines Feldes gibt man an, aus wie vielen Variablen es bestehen soll. Das im Bild gezeigte Feld mit vier Variablen wird so erzeugt:

```
int[] Zahl = new int[4];
```

Mit einer for-Schleife kann man der Reihe nach alle Feldvariablen mit einer Wertzuweisung initialisieren bzw. die Werte ausgeben:

```
for (int i = 0; i < 4; i++) {
  Zahl[i] = i*i;
  System.out.println(Zahl[i]);
}
```

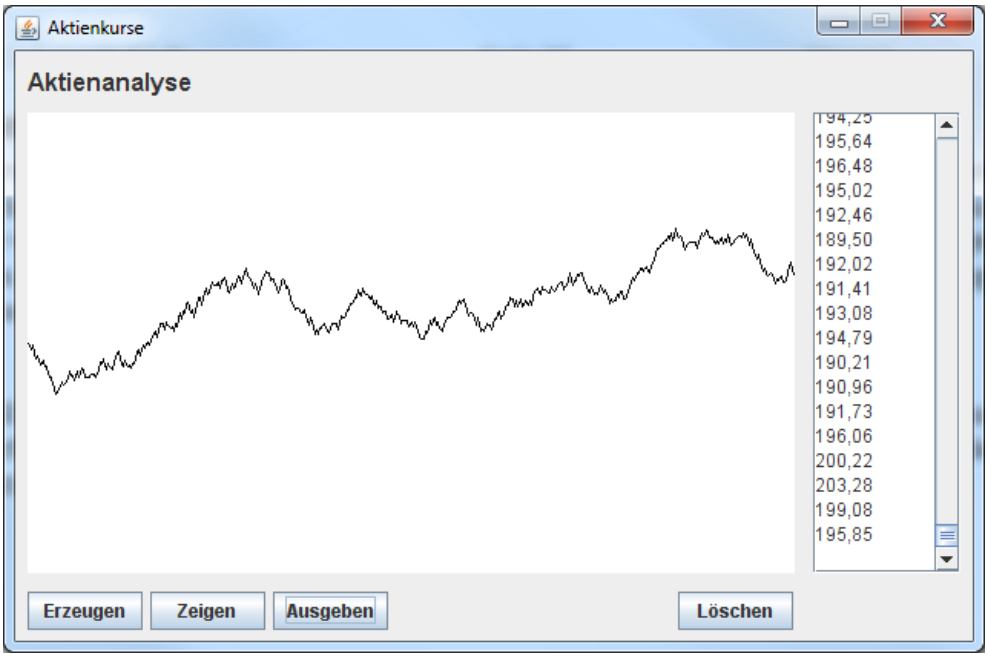
Aufgabe

a) Definiere ein Feld *Kurse* für 500 double-Werte.

b) Speichere mit Hilfe einer Schleife den Wert 500-i in Kurse[i].

Aktienanalyse

Wer an der Börse erfolgreich sein will, der muss sich mit Aktienkursen auseinander setzen. Die technische Analyse geht von historischen Aktienkursen aus und versucht daraus Rückschlüsse auf die weitere Kursentwicklung zu ziehen, um damit Kauf- oder Verkaufsentscheidungen zu begründen. Die Kursentwicklung einer Aktie ergibt sich aus den Tageskursen über einen längeren Zeitraum.



Soll die Kursentwicklung mit Hilfe des Computers analysiert werden, so muss jeder einzelne Tageskurs als Wert verfügbar sein. Für diesen Zweck kann man ein Feld einsetzen, indem für jeden Tag der Kurs einer Aktie gespeichert wird. Die Deklaration des Feldes kann so erfolgen:

```
int Werte = 500;                    vor // Anfang Methoden eingeben
int Startkurs = 150;
double[] Kurse = new double[Werte];
```

In diesem Beispiel können 500 Tageskurse im Feld gespeichert werden. Da wir keine 500 Werte eingeben wollen, erzeugen wir diese zufällig.

Aufgabe

Analysiere und erprobe diese Prozedur:

```
public void bErzeugen_ActionPerformed(ActionEvent evt) {
  double Kurs = Startkurs;
  double Aenderung;
  for (int i = 0; i < Werte; i++) {
    Kurse[i] = Kurs;
    Aenderung = Math.random()*10 - 5;
    Kurs = Kurs + Aenderung;
    System.out.println(Kurse[i]);
  } // end of for
} // end of bErzeugen_ActionPerformed
```

Mit 500 Tageskursen kann man als Mensch wenig anfangen. Also muss der Computer die Daten grafisch darstellen. Wir verwenden dazu die Turtle-Komponente.

Aufgabe

a) Analysiere und erprobe diese Prozedur, wobei davon ausgegangen wird, dass die Zeichenfläche der Turtle 300 x 500 Pixel groß ist.

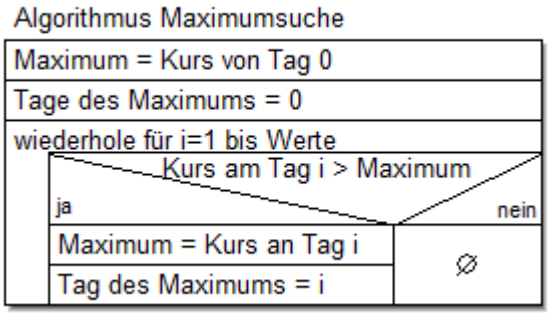
```
public void bZeigen_ActionPerformed(ActionEvent evt) {
  turtle1.setOrigin(0, 300);
  turtle1.moveto(0, Startkurs);
  double y;
  for (int i = 0; i < Werte; i++) {
    y = Kurse[i];
    turtle1.drawto(i, y);
  } // end of for
} // end of bZeigen_ActionPerformed
```

b) Programmiere eine Prozedur, die die Kurswerte der Aktie als Zahlen in einer TextArea-Komponente ausgibt. Die Ausgabe erfolgt mit einer for-Schleife über
`taAusgabe.append(InOut.format2(Kurse[i]) + "\n");`

Minimum und Maximum

Zur Analyse von Aktienkursen gehört auch die Bestimmung des niedrigsten und höchsten Aktienkurses im betrachteten Zeitraum. Doch wie kann man das Minimum bzw. Maximum von 500 Aktienwerten bestimmen? Betrachten wir einen Algorithmus zur Suche des Maximums.

Wir beginnen die Suche am Tag 0 und speichern den zugehörigen Kurs der Aktie als momentanes Maximum. In einer Schleife vergleichen wir dann den Kurs an Tag i mit dem momentanen Maximum. Ist der Kurs an Tag i größer als das momentane Maximum, so ersetzen wir das momentane Maximum durch den Kurs an Tag i und merken uns zusätzlich die Tagesnummer. Nach Ablauf der Schleife ist das momentane Maximum gleich dem Maximum über den gesamten Zeitraum.



Die Implementierung kann dann so erfolgen:

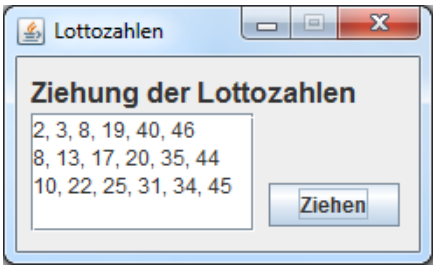
```
public void bMaximum_ActionPerformed(ActionEvent evt) {
  double Maximum = Kurse[0];
  int TagMaximum = 0;
  for (int i = 1; i < Werte; i++) {
    ... // vervollständigen
  } // end of for
  turtle1.setForeground(Color.GREEN);
  turtle1.moveto(0, Maximum);
  turtle1.drawto(Werte, Maximum);
  turtle1.moveto(TagMaximum, 0);
  turtle1.drawto(TagMaximum, Maximum);
} // end of bMaximum_ActionPerformed
```

Gleitender Durchschnitt

Der gleitende Durchschnitt ist ein Mittel der technischen Aktienanalyse, um Kauf- oder Verkaufsignale zu erzeugen. Er besteht aus dem Durchschnitt der letzten 38 (20, 90, 200) Tage. Berechnet man den gleitenden Durchschnitt für jeden Tag und stellt die Durchschnitte grafisch dar, so erhält man einen geglätteten Kursverlauf, der Trends darstellt.

Zur Berechnung des gleitenden Durchschnitts verwendet man ein zweites Feld `double[] Durchschnitte = new double[500];` und berechnet ab dem 37. Tag für jeden Tag den Durchschnitt der vergangenen 38 Tage. Dazu muss man zwei for-Schleifen schachteln. Abschließend zeichnet man die Durchschnittswerte analog zu den Kurswerten mit Hilfe der Turtle.

Lottozahlen



Die Ziehung der Lottozahlen soll simuliert werden. Sechs verschiedene Zahlen zwischen 1 und 49 müssen also zufällig erzeugt werden. Mit unseren bisherigen Mitteln ist das nur sehr schwer zu realisieren. Wir können nicht einfach sechs Zufallszahlen zwischen 1 und 49 erzeugen und ausgeben, denn jede Zufallszahl muss sich von allen vorher gezogenen unterscheiden. Mit Hilfe eines Feldes lassen sich die Probleme leicht lösen.

```
public void bZiehen_ActionPerformed(ActionEvent evt) {
  int[] Kugel = new int[49];

  // Bestimmung der Lottozahlen
  int Zufallszahl;
  for (int i = 1; i <= 6; i++) {
    do {
      Zufallszahl = (int) (Math.random()*48) + 1;
    } while (Kugel[Zufallszahl] != 0);
    Kugel[Zufallszahl] = i;
  } // end of for

  // Ausgabe der Lottozahlen
  String Ausgabe = "";
  for (int i = 0; i < 49; i++) {
    if (Kugel[i] > 0)
      Ausgabe = Ausgabe + ", " + i;
  } // end of for
  Ausgabe = Ausgabe.substring(2);
  taAusgabe.append(Ausgabe + "\n");

} // end of bZiehen_ActionPerformed
```

Aufgaben

a) Erstelle das Programm Lottozahlen.
b) Analysiere die Funktionsweise des Programms im Einzelschrittmodus.
c) Stelle das Feld Kugel in einer Tabelle (Excel/Word) dar, nachdem eine Ziehung erfolgt ist.
d) Erläutere, wie das Feld benutzt wird, um die Ziehung der Lottozahlen zu simulieren.
e) Erweitere das Programm so, dass noch die Zusatzzahl gezogen und ausgegeben wird.
f) Ändere das Programm so ab, dass die Lottozahlen in der Reihenfolge der Ziehung ausgegeben werden.
g) Stelle das Programm als Struktogramm dar.
h) Ändere das Programm so ab, dass statt einer do-while-Schleife eine while-Schleife benutzt wird und vergleiche die beiden Lösungen.

 Gerhard Röhner, September 2015