

Objektorientierte Programmierung

- Einleitung
- Strukturierte Programmierung
- Grundbegriffe der objektorientierten Programmierung
- Arbeiten mit Klassen-Bibliotheken
- Programmieren graphischer Oberflächen
- Aufgaben
- Anhang

Inhaltsverzeichnis

Übersicht

- Einleitung
 - Programmierung
 - Algorithmen
 - Grundeigenschaften von Java
 - Erste Schritte mit Java
- Strukturierte Programmierung
 - Einführung
 - Programmaufbau
 - Datentypen
 - Allgemeines
 - Einfache Datentypen
 - Arrays
 - Zeichenketten
 - Kontrollstrukturen
 - Unterprogramme
- Grundbegriffe der objektorientierten Programmierung
 - Objekte und Klassen
 - Erzeugung von Objekten
 - Verwendung von Objekten
 - Zugriffskontrolle
 - Klassenvariablen und -methoden
 - Vererbung
 - Polymorphie
- Arbeiten mit Klassen-Bibliotheken
 - Pakete
 - Grundlegende Klassen
 - Fehlerbehandlung
 - Ein- und Ausgabe
- Programmieren graphischer Oberflächen
 - Applets
 - Dialogelemente
 - Einfache Graphik
 - Reagieren auf Ereignisse
 - Anordnung der Komponenten
- Aufgaben
 - Aufgabe 1
 - Applet zu Aufgabe 1
 - Source zu Aufgabe 1
 - Aufgabe 2
 - Applet zu Aufgabe 2
 - Source zu Aufgabe 2
 - Aufgabe 3
 - Applet zu Aufgabe 3

- Source zu Aufgabe 3
- Aufgabe 4
 - Applet zu Aufgabe 4
 - Source zu Aufgabe 4
- Aufgabe 5
 - Applet zu Aufgabe 5
 - Source zu Aufgabe 5
- Aufgabe 5a
 - Applet zu Aufgabe 5a
 - Source zu Aufgabe 5a
- Aufgabe 6
 - Applet a zu Aufgabe 6
 - Applet b zu Aufgabe 6
 - Source a zu Aufgabe 6
 - Source b zu Aufgabe 6
- Aufgabe 7
 - Applet zu Aufgabe 7
 - Source a zu Aufgabe 7
 - Source b zu Aufgabe 7
 - Source c zu Aufgabe 7
- Aufgabe 8
 - Applet zu Aufgabe 8
 - Source zu Aufgabe 8
- Aufgabe 8a
 - Applet zu Aufgabe 8a
 - Source zu Aufgabe 8a
- Aufgabe 9
 - Applet zu Aufgabe 9
 - weitere Hinweise zu Aufgabe 9
 - Source a zu Aufgabe 9
 - Source b zu Aufgabe 9
 - Source c zu Aufgabe 9
 - Source d zu Aufgabe 9
 - Source e zu Aufgabe 9
 - Source f zu Aufgabe 9
 - Source g zu Aufgabe 9
 - Source h zu Aufgabe 9
- Aufgabe 10
 - Source zu Aufgabe 10
- Aufgabe 11
 - Applet zu Aufgabe 11
 - Source a zu Aufgabe 11
 - Source b zu Aufgabe 11
 - Source c zu Aufgabe 11
 - Source d zu Aufgabe 11
 - Source e zu Aufgabe 11
 - Source f zu Aufgabe 11

- Source g zu Aufgabe 11
 - Source h zu Aufgabe 11
 - Source i zu Aufgabe 11
- Aufgabe 12
 - Applet zu Aufgabe 12
- Aufgabe 13
 - Applet zu Aufgabe 13
- Anhang
 - Arbeiten mit NetBeans
 - Prüfungsleistung
 - Literatur
 - Sourcen
 - Primzahl.java
 - CopyFile.java
 - HalloApplet.java
 - ZeichenApplet.java
 - EventApplet.java
 - GridLayoutApplet.java
 - BorderLayoutApplet.java
 - BorderLayoutWithBorderApplet.java
 - NiceLayoutApplet.java
 - Applets
 - Primzahl
 - HalloApplet
 - ZeichenApplet
 - EventApplet
 - GridLayoutApplet
 - BorderLayoutApplet
 - BorderLayoutWithBorderApplet
 - NiceLayoutApplet

Einleitung

- Programmierung
- Algorithmen
- Grundeigenschaften von Java
- Erste Schritte mit Java

Programmierung

- Computer - ein Universalwerkzeug:
 - Die Stärke eines Computers ist seine Vielseitigkeit: Er kann programmiert werden, d.h. seine einfachen Grundbefehle können zur Erledigung verschiedenster Aufgaben kombiniert werden.
- Beispiele für Grundbefehle:
 - Verschieben von Daten
 - Bringe den Inhalt von Speicherzelle 4228 in das Register r3
 - Berechnen
 - Addiere den Inhalt der Register r1 und r2 und speichere das Ergebnis in Register r3
 - Steuern des Programmablaufs
 - Hole das nächste Kommando aus der Speicherzelle 2001, falls der Inhalt von Register r1 Null ist.
- Programm:
 - Folge von Kommandos, die von einem Computer ausgeführt werden können
- Maschinensprache:
 - Menge der grundlegenden Befehle, die ein Computer direkt versteht
 - unterschiedlich für verschiedene Rechnerarten bzw. Prozessoren
 - bezieht sich auf die interne Struktur einer CPU (Register, Arithmetik-Einheit etc.)
 - schwer zu benutzen zur Lösung echter Probleme
- Beispiele für "echte" Probleme:
 - Berechne näherungsweise den Wert des Integrals einer gegebenen Funktion zwischen bestimmten Grenzen.
 - Untersuche einen menschlichen Satz, um festzustellen, welche Anfrage an ein Flugbuchungssystem gestellt wurde.
 - Finde alle Personen aus einer großen Menge von Bankkonten-Daten, deren Konto um einen bestimmten Betrag überzogen ist.
- Höhere Programmiersprachen:
 - verwenden Befehle aus einem bestimmten Anwendungsgebiet
 - abstrahieren von der Prozessor-Hardware
 - sind leichter zu benutzen
- Beispiele:
 - Mathematische Abstraktionen sind die Domäne von Fortran. Es erlaubt, eine mathematische Formel wie
 - $$x_1 = -\frac{p}{2} + \sqrt{\frac{p^2}{4} - q}$$
 - in der folgenden einfachen Weise zu formulieren:
 - $x_1 = -p/2 + \text{sqrt}(p**2/4 - q)$
 - Logische Abhängigkeiten und Beziehungen bilden die Basiskommandos von Prolog. Damit kann man leicht Dinge formulieren wie
 - Satz = (Subjekt + Verb) ODER (Subjekt + Verb + Objekt)
 - Spezielle Datenbank-Sprachen wie SQL erlauben direkt Abfragen von Daten, z.B.:
 - Zeige alle NAME Felder im Datensatz KONTEN für die (KONTOSTAND < -100 Euro)

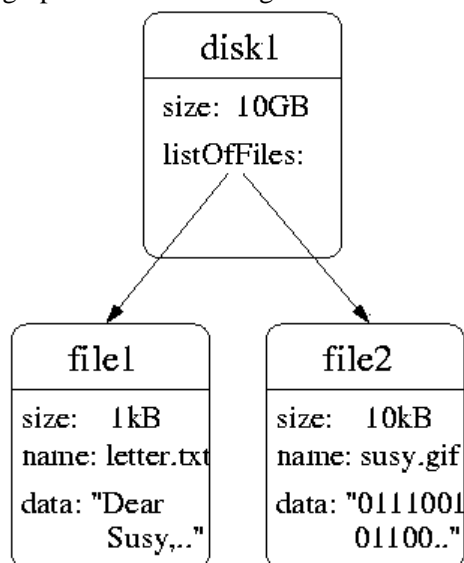
- Problem:
 - Programme in solchen Sprachen müssen in die Maschinensprache des Prozessors übersetzt werden, damit sie ausgeführt werden können. Diese mühsame Arbeit kann selbst wieder von einem Programm erledigt werden.
- Compiler:
 - Programm, das einen Programmtext einliest und in Maschinensprache übersetzt.

Algorithmen

- Algorithmus:
 - Beschreibung eines allgemeinen Verfahrens zur Lösung eines gegebenen Problems mit Hilfe einfacher Schritte
 - soll immer in endlich vielen Schritten fertig sein
 - allgemein beschreibbar, unabhängig von einer Programmiersprache
 - meistens für (potentiell unendlich viele) verschiedene Eingabewerte
 - hier nur Algorithmen, die auf Computern programmiert werden können
- Beispiel:
 - Addition zweier großer Zahlen
 - Beschreibung
 - schreibe beide Zahlen stellenrichtig untereinander
 - addiere die Einerstellen
 - Ergebnis $< 10 \rightarrow$
schreibe das Ergebnis als letzte Stelle der Summe
 - Ergebnis $\geq 10 \rightarrow$
schreibe Einer des Ergebnisses als letzte Stelle der Summe
markiere 1 als Übertrag in der vorletzten Stelle
 - Gehe analog für die nächsten Stellen vor, wobei ggf. noch ein Übertrag aufaddiert wird
 - Ein Übertrag in der ersten Stelle wird als 1 vor das Ergebnis geschrieben
 - Was sind "einfache Schritte"? Hier z.B. Addition von zwei einstelligigen Zahlen
 - Zahl der Schritte hängt von der Eingabe (Stellenanzahl der Summanden) ab
- Vergleich von Algorithmen:
 - häufig verschiedene Algorithmen zur Lösung eines Problems
 - Vergleich der Algorithmen etwa nach Umsetzung in Programm
 - Unterschiede z.B. im Zeitaufwand (Rechendauer), benötigtem Speicher, Programmieraufwand
- Einsatz von Algorithmen:
 - für viele wichtige Probleme gute (schnelle, speichereffiziente, ...) Algorithmen vorhanden
 - eigenen ad-hoc-Lösungen fast immer überlegen
 - Beispiele:
 - schnelle Sortierverfahren wie Quicksort
 - Zeichenverfahren auf Bildschirmen (Pixelgraphik)
 - schnelle Fourier-Transformation (FFT)
 - häufig basieren bessere Algorithmen auf neuen mathematischen Einsichten
 - Merke: Ein Auto kaufen führt meist schneller zum Ziel als das Rad neu erfinden

Grundeigenschaften von Java

- Java:
 - objektorientierte Programmiersprache
 - klares Design
 - wichtige Programmiersprache für Internet-Anwendungen
 - weit verbreitet
 - entwickelt von der Firma Sun Microsystems
- Objektorientierte Sprachen:
 - Gruppe von momentan sehr populären Programmiersprachen
 - erlauben die Konstruktion von Programmen aus kleineren Bausteinen (den Objekten)
 - erhöhen Übersichtlichkeit und Wiederverwendbarkeit
- Objekt:
 - besteht aus Datenfeldern und Methoden
 - Datenfelder
 - Variablen für dieses Objekt
 - beinhalten Zustand des Objekts
 - in der Regel nicht direkt zugreifbar
 - Methoden
 - beschreiben mögliche Aktionen mit den Daten
 - Verhalten des Objekts
 - erlauben kontrollierten Zugriff auf Daten
- Beispiel:
 - Betriebssystem organisiert Daten auf einer Festplatte
 - Datei-Objekte (file1, file2)
 - Felder size, name, data
 - Methoden getSize, appendData(newData)
 - Festplatten-Objekt disk1
 - Felder: size, listOfFiles
 - Methoden: createNewFile(name), getFreeSpace
 - graphische Darstellung



- Klasse:
 - Beschreibung aller Objekte mit gleichen Datenfeldern und Methoden
 - Vorlage zum Erstellen neuer Objekte
 - im Beispiel
 - Objekte file1 und file2 gehören zur gemeinsamen Klasse File
 - Objekt disk1 gehört zur Klasse Disk
- Übersetzen eines Java-Programms:
 - mit dem Java-Compiler javac
 - javac STARTKLASSE.java
 - erzeugt Dateien KLASSE.class für jede benötigte Klasse
 - findet und übersetzt selbsttätig Hilfsklassen
 - übersetzt nicht in Maschinsprache, sondern in universellen Java-Bytecode
- Ausführen eines Java-Programms:
 - mit dem Programm java (virtuelle Java-Maschine = JVM)
 - java STARTKLASSE
 - interpretiert Bytecode, erzeugt Maschinencode und führt ihn aus
 - Vorteil: Derselbe Bytecode läuft unverändert auf allen Maschinen, die eine JVM haben
 - Wichtig für Internetanwendungen: läuft auf allen Clients
 - Eine JVM ist in der Regel in Web-Browser integriert (für Applets)

Erste Schritte mit Java

- Primzahl - unser erstes Java-Programm:
 - prüft, ob die eingegeben Zahl eine Primzahl ist
 - gibt ggf. einen Teiler aus
 - vollständiger Code in Primzahl.java
 - Name der Datei = Name der Startklasse + Endung .java
 - als lauffähiges Applet
 - i.f. grobe Übersicht, detaillierte Erklärungen im Lauf des Kurses
- Funktionsweise des Programms:
 - Algorithmus
 - Teste, ob zahl durch 2 teilbar ist
 - Teste, ob zahl durch ungerade Zahlen 3, 5, 7 ... max teilbar ist
 - maximaler Testwert ist Wurzel aus zahl
 - Probleme
 - ungültige Eingaben
 - falsche Ergebnisse bei 1 und 2
- Grobstruktur:
 - import java.io.*;
 - Angabe im Programm benutzter Standard-Javaklassen
 - hier für Ein- und Ausgabe
 - public class Primzahl { ... }
 - jedes Java-Programm besteht aus Klassen
 - öffentlich verwendbare Klassen definiert mit
 - public class NAME
 - in einer Datei immer nur eine öffentliche Klasse
 - Name der Datei muss NAME.java sein
 - public static void main(String[] args)
 - Standardname für das "Hauptprogramm"
 - Klassen mit main-Routine können mit java-Programm gestartet werden
 - Argument args für zusätzliche Parameter auf der Kommandozeile
 - i.f. nicht weiter verwendet
- Textanordnung eines Java-Programms:
 - beliebige Aufteilung in Zeilen (solange man kein Wort trennt)
 - beliebig viele Leerräume zwischen Wörtern
 - Ende einer Anweisung mit ; gekennzeichnet
 - Einrückung der Zeilen
 - systematisch, um die Struktur des Programms zu verdeutlichen
 - allgemeine Konvention, kein Sprachelement
 - wichtige Hilfe für den (menschlichen) Leser
 - Kommentar
 - erläuternder Text für den Leser
 - kein Teil des eigentlichen Programms
 - von // bis zum Zeilenende
- Ein- und Ausgabe:
 - vordefinierte Objekte System.out und System.in
 - Ausgabe einer Textzeile mit

- `System.out.println("Text");`
 - Einlesen einer Textzeile in Zeichenkette `s` mit
 - `s = in.readLine();`
 - zum Einlesen wird `BufferedReader` benötigt
 - mögliche Fehler beim Einlesen angezeigt durch
 - `throws IOException`
- Variablen:
 - Zwischenspeicher für Werte
 - unterschiedliche Typen, z.B.
 - `int` ganze Zahl
 - `double` Fließkomma-Zahl
 - `String` Zeichenkette
 - `boolean` ja/nein-Entscheidung (`true/false`)
 - bekommen Werte mit `=`
- Operationen mit Variablen:
 - können mit üblichen Operationen (`+`, `-`, `*`, `/`) verknüpft werden
 - Rest bei Division mit `%` (Modulo-Operation)
 - können mit üblichen Operationen (`==`, `!=`, `<`, `<=`) verglichen werden
 - spezielle mathematische Funktionen in Klasse `Math`
 - `sqrt` Quadratwurzel
 - `floor` ganzzahliger Anteil (nach unten gerundet)
 - Umwandlung zwischen Typen
 - `i = (int) d;` von `double` nach `int`
 - `i = Integer.parseInt(s);` von `String` nach `int`
- Kontrollstrukturen:
 - beeinflussen Reihenfolge der Anweisungen
 - Alternative
 - ```

 if (BEDINGUNG) {
 ANWEISUNGEN1
 } else {
 ANWEISUNGEN2
 }

```

      - abhängig von `BEDINGUNG` wird die eine oder andere Anweisungsgruppe ausgeführt
  - Zählschleife
    -

```
for (int i=3; i<=max; i=i+2) {
 ANWEISUNGEN
}
```

- ANWEISUNGEN werden wiederholt, solange  $i \leq \text{max}$
- Startwert  $i=3$
- bei jeder Wiederholung wird  $i$  um 2 erhöht
- Aufgaben:
  - Aufgabe 1
  - Aufgabe 2

# Strukturierte Programmierung

- Einführung
- Programmaufbau
- Datentypen
- Kontrollstrukturen
- Unterprogramme

# Einführung

- Programme unverständlich durch
  - verworrene Ablauf-Reihenfolge mit goto-Anweisung ("Spaghetti-Programmierung")
  - in maschinennaher Weise abgelegte Daten ("Speicherplatz ausnutzen")
  - große, undurchschaubare Programmteile
- Lösungsansatz der strukturierten Programmierung:
  - Datenstrukturen, die den mit den Daten durchzuführenden Operationen entsprechen
  - Kontrollstrukturen, die den Programmablauf deutlich werden lassen (kein goto mehr)
  - Techniken zur Zerlegung der Programme in kleine Einheiten (Unterprogramme), die weitgehend unabhängig voneinander entwickelt werden können
- Programmiersprachen:
  - PASCAL als Prototyp: schön, aber vom Aussterben bedroht (Ausnahme: Delphi und Nachfolger)
  - Ideen in vielen anderen Sprachen übernommen (z.B. C, C++, Java)

# Programmaufbau

- Java-Applikation:
  - selbständiges Programm
  - wird von der Kommandozeile oder Bediener-Oberfläche aus gestartet
  - muss eine öffentliche Klasse mit main-Methode enthalten der Form
    - `public static void main (String[] args)`
- Java-Applet:
  - Programmteil, das innerhalb eines Web-Browsers abläuft
  - enthält keine main-Methode (steckt im Browser)
  - direkt aus dem Internet ladbar
  - in eine HTML-Seite eingebettet
  - Möglichkeiten stark eingeschränkt (z.B. kein Zugriff auf lokale Dateien)
- Programm-Aufteilung:
  - eine Datei für jede öffentliche Klasse
  - Name der Datei `KLASSENNAME.java` ist zwingend vorgeschrieben
- Namen (für Variablen, Klassen etc.):
  - dürfen nur aus Buchstaben und Ziffern bestehen
  - auch Umlaute und Buchstaben anderer Schriften erlaubt
  - Zeichen `_` und `$` zählen in Java zu den Buchstaben!
  - erstes Zeichen muss ein Buchstabe sein
  - gültige Namen: `meinName`, `__intern__`, `Größe45`, `Æµç`
- Namens-Konventionen:
  - Vereinbarungen zur Verbesserung der Lesbarkeit, "freiwillig"
  - in Kleinbuchstaben: Variablen und Methoden
  - erster Buchstabe groß, sonst klein: Klassen
  - in Großbuchstaben: symbolische Konstanten



# Datentypen

- Allgemeines
- Einfache Datentypen
- Arrays
- Zeichenketten

# Allgemeines

- Datentypen:
  - beschreiben die Menge möglicher Werte
  - legen die möglichen Operationen fest
  - in Java: bestimmen den benötigten Speicherplatz und die Interpretation des Bitmusters
- Beispiel byte:
  - beschreibt ganze Zahlen von -128 .. 127
  - arithmetische Operationen +, -, \*, /, %
  - benötigt 8 Bit (= 1 Byte)
  - Interpretation im Zweierkomplement:
    -

|       |        |       |       |       |       |       |       |       |       |
|-------|--------|-------|-------|-------|-------|-------|-------|-------|-------|
|       | $-2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |       |
|       | 1      | 0     | 1     | 0     | 0     | 1     | 1     | 1     |       |
| Wert: | -128   | +0    | +32   | +0    | +0    | +4    | +2    | +1    | = -89 |

- Variable:
  - benannte Speicherstelle zum Ablegen und Zugreifen auf Daten
  - hat einen Namen und einen Datentyp
  - muss spätestens beim ersten Auftreten deklariert (d.h. Name und Typ vereinbart) werden, z.B.:

```
double a;
```

- Variablendeklarationen sind Anweisungen (sie können insbesondere überall im Code stehen)

- Einfache Datentypen:
  - in Java vordefinierte Typen für Zahlen und Zeichen
  - Variablen enthalten dem Datentyp entsprechende Werte
  - Zuweisung mit = bewirkt eine Kopie des Wertes

- vorher

|   |
|---|
| 3 |
| a |

|   |
|---|
| 0 |
| b |

- 

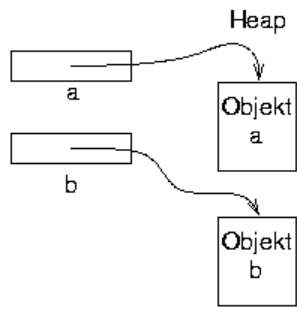
- nach `b = a`

|   |
|---|
| 3 |
| a |

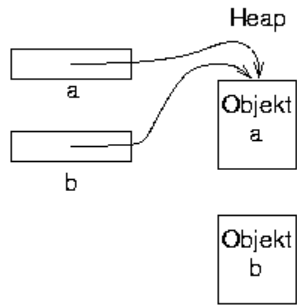
|   |
|---|
| 3 |
| b |

- 

- Höhere Datentypen (Referenztypen):
  - fassen mehrere Daten zu einem Typ zusammen
  - in Java Felder (Arrays) und Klassen
  - Variablen dieser Typen enthalten nur Zeiger (Referenzen) auf die Objekte
  - Heap = Speicherbereich für alle Objekte
  - Zuweisung mit = bewirkt eine Kopie des Zeigers
    - vorher



- 
- nach  $b = a$



-

# Einfache Datentypen

- Übersicht:

| Datentyp | Bedeutung            | Größe in Byte |
|----------|----------------------|---------------|
| byte     | ganze Zahl           | 1             |
| short    | ganze Zahl           | 2             |
| int      | ganze Zahl           | 4             |
| long     | ganze Zahl           | 8             |
| char     | Zeichen (im Unicode) | 2             |
| float    | Fließkommazahl       | 4             |
| double   | Fließkommazahl       | 8             |
| boolean  | logischer Wert       | -             |

- Konstanten:

- feste Werte, die direkt im Programmtext stehen, oder mit Namen versehen
- benannte Konstanten mit Schlüsselwort final, z.B.

```
final int MAX_VALUE = 42;
```

- angehängtes L oder l: vom Typ long
- angehängtes F oder f: vom Typ float
- vorangestelltes 0x oder 0X: ganze Zahlen im Hexadezimalsystem
- Beispiele:

| Typ     | Beispiele            |
|---------|----------------------|
| int     | 28, -12345, 0xFF     |
| long    | 1234567910L          |
| float   | 3.1415926f           |
| double  | 2.718281828, 1.4e-45 |
| boolean | true, false          |
| char    | 'a', 'ö', '\n'       |

- Darstellung von Fließkomma-Zahlen:

- aufgeteilt in Vorzeichen, Mantisse (Ziffernfolge) und Exponent (zur Basis 2):

| Typ    | Bits Mantisse | Bits Exponent | Dezimalen | Wertebereich   |
|--------|---------------|---------------|-----------|----------------|
| float  | 23            | 8             | 7         | $10^{\pm 38}$  |
| double | 52            | 11            | 15        | $10^{\pm 308}$ |

- spezielle Werte (jeweils für float und double)

| Wert                 | Beispiel | symbolische Konstante (analog für float) |
|----------------------|----------|------------------------------------------|
| unendlich            | 1.0/0.0  | Double.POSITIVE_INFINITY                 |
| -unendlich           | -1.0/0.0 | Double.NEGATIVE_INFINITY                 |
| NaN ("Not a Number") | 0.0/0.0  | Double.NaN                               |

- grundlegende Operatoren:
  - arithmetisch: +, -, \*, /, %  
Grundrechenarten und Modulo
  - Vergleichsoperatoren: ==, !=, <, <=, >, >=  
Ergebnis ist vom Typ boolean
  - logische Operatoren: && (und), || (oder), ! (nicht)  
verknüpfen Werte vom Typ boolean
  - nützliche Abkürzungen:

| Operation | Bedeutung   |
|-----------|-------------|
| a++;      | a = a + 1;  |
| a--;      | a = a - 1;  |
| a += 10;  | a = a + 10; |

analog für die anderen binären Operatoren

- Beispiele:
  -

| Ausdruck             | Wert |
|----------------------|------|
| 12 % 5               | 2    |
| -12 % 5              | -2   |
| 5 < 8                | true |
| (5 == 12)    (5 < 8) | true |

- Zuweisung zwischen unterschiedlichen numerischen Typen:
  - von "kleineren" zu "umfassenderen" Typen automatisch
  - umgekehrte Richtung nur mit explizitem Umwandlungsoperator (cast)
    - (NEUERTYP) Variable
  - Beispiele:

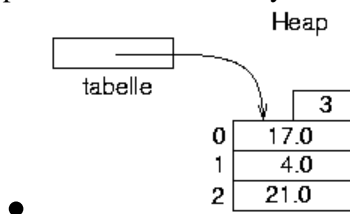


```
int i = 17;
long l = 1000L;
l = i; // ok, kein Verlust
i = l; // Fehlermeldung vom Compiler!
i = (int) l; // ok, wenn man weiss, was man tut
byte b = (byte) 256; // ergibt 0!
```

- Aufgaben:
  - Aufgabe 3

# Arrays

- Feld (Array):
  - Referenztyp, der mehrere Elemente eines Grundtyps zusammenfasst
  - wird vereinbart (Deklaration) mit
    - GRUNDTYP[] ARRAYNAME;
  - Beispiel
    - double[] tabelle;
  - Festlegung der Größe und Besorgen des Speicherplatzes (Definition) mit new
    - tabelle = new double[3];
  - Deklaration und Definition auch in einem Schritt möglich
    - double[] tabelle = new double[3];
- Elemente eines Arrays:
  - Zugriff mit Index i
    - ```
tabelle[0] = 17.0;
tabelle[1] = 4.0;
tabelle[2] = tabelle[0] + tabelle[1];
```
 - Bereich für Index $0 \leq i \leq \text{length} - 1$
 - Anzahl im Array als Datenfeld length abgelegt
 - int maxIndex = tabelle.length - 1;
 - bei Index-Überschreitung Abbruch mit Fehlermeldung (IndexOutOfBoundsException)
 - Initialisierung mit Liste von Anfangswerten
 - int[] lotto = {2, 11, 12, 14, 34, 36};
- Speicherplatz bei Referenztypen:
 - zeigen nach der Deklaration nirgendwohin (Wert null)
 - neue Objekte mit new, d.h. es wird Speicherplatz für sie reserviert (im Heap).
 - nicht mehr gebrauchte Objekte (d.h. keine Variable zeigt darauf) werden automatisch gelöscht
 - Speicherabbild des Arrays tabelle

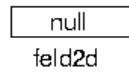


- Mehrdimensionale Arrays:
 - Arrays, deren Elemente selbst Arrays sind
 - Beispiel:
 -

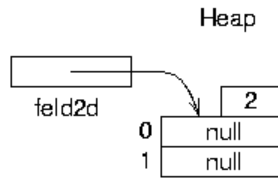
```
double[][] matrix = new double[100][100];
matrix[3][4] = 7.0;
```

- Zeilenarrays müssen nicht gleich lang sein
- Beispiel:
 -

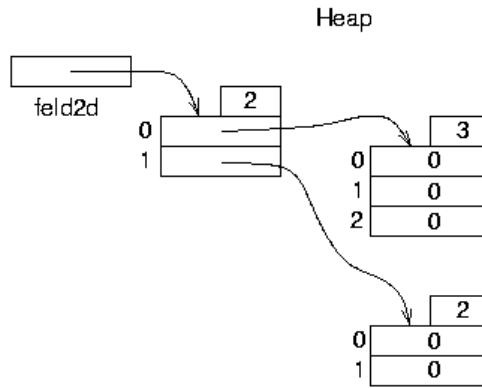
`int[][] feld2d;`



`feld2d = new int[2][];`



`feld2d[0] = new int[3];`
`feld2d[1] = new int[2];`



Zeichenketten

- Zeichenkette:
 - Objekt der Standard-Klasse String
 - konstanter String: Zeichenfolge, eingeschlossen in "
 - String s1 = "juhu";
 - Verkettung von Strings mit +
 - String s2 = "Peter" + " " + "Junglas";
- Methode toString:
 - für alle Standardtypen definiert
 - erzeugt eine Stringdarstellung eines Objekts
 - wird bei Verkettung vom Compiler automatisch hinzugefügt
 -

```
String str    = "Ergebnis: ";  
double result = 47.11;  
System.out.println(str + result);  
// Ausgabe: "Ergebnis: 47.11"
```

- Einige Methoden von String:

Methode	Bedeutung
String(char[] value)	erzeugt neuen String aus char-Array
int length()	Zahl der Zeichen des Strings
char charAt(int index)	Zeichen an der Position index
String substring(int begin, int end)	Teilstring s(begin, end - 1)
int compareTo(String s2)	vergleicht mit String s2, 0 bei Gleichheit

Kontrollstrukturen

- Steuerung des Programmablaufs:
 - verschiedene Arten von Verzweigungen und Schleifen
 - Grundeigenschaft: Blöcke mit einem Startpunkt und einem Endpunkt
 - keine Sprünge an andere Stellen des Programms (goto)
 - Problem Fehlerbehandlung: Ausnahmen (s.u.)

- Block:

- Syntax:

```
{
    Anweisung1;
    Anweisung2;
    ...
    Anweisungn;
}
```

- gilt als eine Anweisung

- Bedingte Anweisung:

- Syntax:

```
if (boolescher Ausdruck)
    Anweisung1;
else
    Anweisung2;
```

- Ist der boolesche Ausdruck true, wird Anweisung1 ausgeführt, sonst Anweisung2.
- Hier wie im folgenden kann überall eine einzelne Anweisung durch einen Block ersetzt werden.
- Der else-Teil kann entfallen.

- Verzweigung:

- Syntax:

```
switch (Auswahl-Ausdruck) {
    case Konstante1:
        Anweisung1;
        break;
    case Konstante2:
        Anweisung2;
        break;
    ...
    default:
        AnweisungN;
}
```

- Auswahl-Ausdruck ist von einem Integer-Typ.
- Je nach Wert des Auswahl-Ausdrucks wird der entsprechende case-Zweig angesprungen und die zugehörige Anweisung ausgeführt. Das break springt dann hinter die Verzweigung.
- Der default-Zweig kann entfallen. Er wird ausgewählt, wenn der Wert sonst nicht auftaucht.
- Fehlt ein break, werden die Anweisungen des folgenden Zweigs ausgeführt. Dies kann gelegentlich nützlich verwendet werden, aber meistens ist es ein Fehler.

- Abweisende Schleife:

- Syntax:

```
while (boolescher Ausdruck)
    Anweisung;
```

- Anweisung wird solange ausgeführt, bis der boolesche Ausdruck false ist.
- Ist der Ausdruck schon zu Beginn false, wird die Schleife übersprungen.

- Nichtabweisende Schleife:

- Syntax:

```
do
    Anweisung;
while (boolescher Ausdruck);
```

- wie die while-Schleife, aber der Test erfolgt erst am Ende.
- Schleife wird immer mindestens einmal durchlaufen.

- Laufanweisung:

- Syntax:

```
for (Init; boolescher Ausdruck; Inkrement)
    Anweisung;
```

- abkürzende Schreibweise für

```
Init;
while (boolescher Ausdruck) {
    Anweisung;
    Inkrement;
}
```

- hauptsächlich für einfache Zählschleifen verwendet:

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

- continue-Anweisung:

- überspringt den Rest des aktuellen Schleifendurchgangs und startet den nächsten (ggf. incl. Inkrementierung und Test)

- break-Anweisung:

- dient zum sofortigen Verlassen einer Schleife oder Verzweigung
- wird eingesetzt, wenn besondere Bedingung die weitere Schleifenausführung verhindert oder überflüssig macht
- Beispiel:

```
for(int alter = 18; alter <= 65; alter++) {
    boolean gewonnen = spieleLotto();
    if (gewonnen) break;
    arbeite();
}
```

- Aufgaben:

- Aufgabe 4
- Aufgabe 5
- Aufgabe 5a

Unterprogramme

- Unterprogramme oder Routinen:
 - Folgen von Anordnungen, die unter einem Namen aufgerufen werden können
 - in Java immer als Methoden einer Klasse zugeordnet
 - bekommen Eingabedaten durch eine Menge von Variablen (Parameterliste), deren Werte vom Aufrufer übergeben werden
 - geben als Ergebnis einen Wert zurück
 - können darüber hinaus Änderungen durchführen, die auch im aufrufenden Programm sichtbar sein können (Seiteneffekte)
 - können eigene Variablen (lokale Variablen) definieren, die zu Beginn und nach dem Beenden der Routine keinen Wert haben
 - jedes Programm hat eigenen Speicherbereich (Stack) für seine lokalen Variablen

- Beispiel:

- Routine hypot



```
double hypot(double a, double b) {
    double ergebnis = Math.sqrt(a*a + b*b);
    return ergebnis;
}
```

- zwei Parameter a, b als Eingabewerte
- Ergebniswert vom Typ double
- lokale Variable ergebnis
- keine Seiteneffekte

- Parameter:

- werden im Unterprogramm wie lokale Variable benutzt
- aufrufendes Programm ersetzt die Parameter durch Ausdrücke der geeigneten Typen
- Werte werden für die Routine kopiert (call by value)

- Rückgabewert:

- Wert wird zum Aufrufer zurückgegeben mit
 - return AUSDRUCK;
- "Pseudo-Datentyp" void = kein Rückgabewert, zurückkehren mit return;
- Wert kann vom Aufrufer ignoriert werden, z.B.



```
double verdopple(double a) {
    return 2*a;
}
```

```
b = verdopple(4.0);
verdopple(3.2);
```

- Seiteneffekte:

- Änderungen durch Routine im aufrufenden Programm
- Beispiele: Veränderungen an Datenfeldern, an Dateien, an der Systemuhr usw.
- können schwer zu durchschauende Abhängigkeiten bewirken

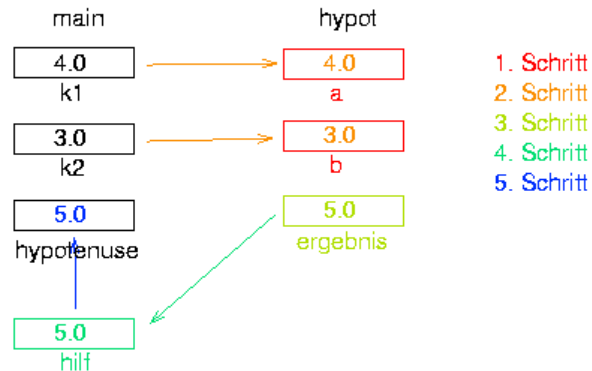
- Ablauf eines Unterprogramms im Detail:

1. Parametervariablen des Unterprogramms werden in seinem Stack angelegt
2. Werte werden vom Aufrufer in die Parametervariablen kopiert

3. Unterprogramm wird ausgeführt
4. beim return wird der Wert des Ergebnisses in Hilfsvariable beim Aufrufer kopiert
5. Programmausführung geht beim Aufrufer weiter

- Beispiel:

- `double k1 = 4.0`
- `double k2 = 3.0;`
- `double hypotenuse;`



`hypotenuse = hypot(k1, k2);`

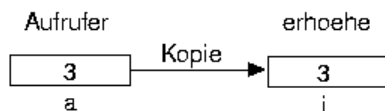
- Parameterübergabe bei einfachem Datentyp:

- Beispiel:

```
void erhoehe(int i) {
    i++;
}

int a = 3;
erhoehe(a);
System.out.println(a); // Ergebnis: 3
```

- keine Änderung von a, da nur der Wert von a kopiert wurde.
- im Bild:



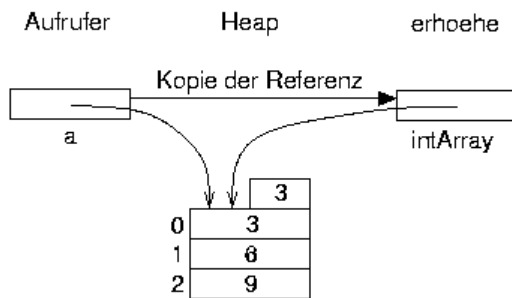
- Parameterübergabe bei Referenztyp:

- Beispiel:

```
void erhoehe(int[] intArray) {
    for (int i=0; i<intArray.length; i++) {
        intArray[i]++;
    }
}

int[] a = {3,6,9};
erhoehe(a);
System.out.println(a[0]); // Ergebnis: 4
```

- keine Änderung an a, nämlich der Referenz, sondern Änderung am Objekt, auf das a verweist.
- im Bild:



- Sichtweisen auf eine Methode:
 - Schnittstelle
 - alles, was von "außen" sichtbar ist
 - Name, Parameterliste und Rückgabewert
 - Implementierung
 - Anweisungen der Methode
 - "Innenleben"
 - Auftrennung erleichtert Programmierung
 - korrekte Funktionsweise (Implementierung) testen
 - danach nur noch Schnittstelle wichtig
- Überladen von Methoden
 - gleicher Name für Methoden mit verschiedenen Parametern
 - Beispiel:
 - ```
double sin(double d)
Complex sin(Complex z)
```
  - Compiler wählt passende Methode nach Parametertypen
- Aufgaben:
  - Aufgabe 6

# **Grundbegriffe der objektorientierten Programmierung**

- Objekte und Klassen
- Erzeugung von Objekten
- Verwendung von Objekten
- Zugriffskontrolle
- Klassenvariablen und -methoden
- Vererbung
- Polymorphie

# Objekte und Klassen

- Objektorientierte Programmierung:
  - Verfahren, auch komplexe Programmsysteme zu bewältigen
  - Grundidee: Zerlegung des Programms in überschaubare Einheiten mit klarer Schnittstelle ("Objekte")
  - Trennung von Schnittstelle und Implementierung
  - Wiederverwendbarkeit der Teile
- Klasse:
  - Referenz-Datentyp, der Datenfelder und die diese manipulierenden Routinen (Methoden) zu einer Einheit zusammenfasst
  - Bauplan für Objekte
  - Datenfelder von beliebigem Typ möglich, auch Arrays und Klassen
  - enthält spezielle Methoden zur Erzeugung von Objekten dieses Typs (Konstruktoren)
- Beispiel:
  -

```
public class Gerade {
 // beschreibt Gerade in zwei Dimensionen

 // Datenfelder
 double m; // Steigung
 double b; // y-Achsenabschnitt

 // Methoden
 // Konstruktor
 Gerade(double m1, double b1) {
 m = m1;
 b = b1;
 }

 void verschiebe(double x, double y) {
 // verschiebt Gerade um Vektor (x, y)
 verschiebeX(x);
 verschiebeY(y);
 }

 public String toString() {
 // Ausgabe: y = m*x + b
 return "y = " + m + "*x + " + b;
 }

 void verschiebeX(double x) {
 // verschiebt Gerade um x in x-Richtung
 b = b - m*x;
 }

 void verschiebeY(double y) {
 // verschiebt Gerade um y in y-Richtung
 b = b + y;
 }
}
```

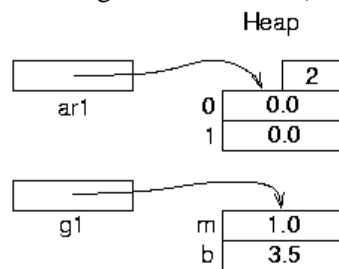
- Konstruktor Gerade erzeugt neues Objekt aus Werten für Steigung und y-Achsenabschnitt
- verschiebe ändert das aktuelle Objekt durch Translation mit dem Vektor (x, y)
- toString erzeugt einen String, der das Objekt repräsentiert, etwa zur Ausgabe



- Hilfsmethoden verschiebeX, verschiebeY
  - werden zur Implementierung von verschiebe benutzt
  - verändern die Datenfelder des Objekts
- Objekt:
  - eine Variable eines Klassentyps
  - hat einen Zustand (= Werte für die internen Datenfelder)
  - stellt Methoden zur Verfügung
  - Deklaration mit Klassennamen
    - Gerade g;
  - Definition (Speicherreservierung) und Initialisierung (Belegung der Datenfelder) mit new und Konstruktor
    - g = new Gerade(2.1, -0.3);
  - muss nicht explizit vernichtet werden - das erledigt das Java-Laufzeitsystem selbst, sobald ein Objekt nicht mehr benutzt wird (garbage collection).

- Vergleich zwischen Arrays und Klassen:

- konkret seien ein Array und eine Gerade gegeben durch
  - double[] ar1 = new double[2];
  - Gerade g1 = new Gerade(1.0, 3.5);



- ar1 und g1 sind beide Referenzvariablen
- Erzeugung
  - jeweils mit new im Heap
  - bei Array durch Angabe der Anzahl von Elementen
  - bei Gerade durch selbstgeschriebenen Konstruktor
- Datenfelder
  - bei ar1 durchnummerierte Felder (ar1[0], ar1[1]) und ar1.length
  - bei g1 selbstdefinierte Felder g1.m, g1.b
- Methoden
  - bei ar1 nur einige Standardmethoden (u.a. toString())
  - bei g1 zusätzlich verschiebe(x, y) etc, beliebig erweiterbar

# Erzeugung von Objekten

- Konstruktor:

- spezielle Methode zur Initialisierung der Datenfelder eines Objekts
- hat den Namen der Klasse, keinen Rückgabewert (auch nicht void)
- nicht-initialisierte Datenfelder setzt der Compiler auf Null
- man kann mehrere Konstruktoren mit unterschiedlichen Parametern definieren (Überladen von Methoden), z.B.:

- 

```

Gerade(double m1, double b1) {
 m = m1;
 b = b1;
}

Gerade(double m1) {
 // erzeugt Gerade durch Ursprung
 m = m1;
 b = 0.0;
}

```

- Ablauf eines Konstruktors im Detail:

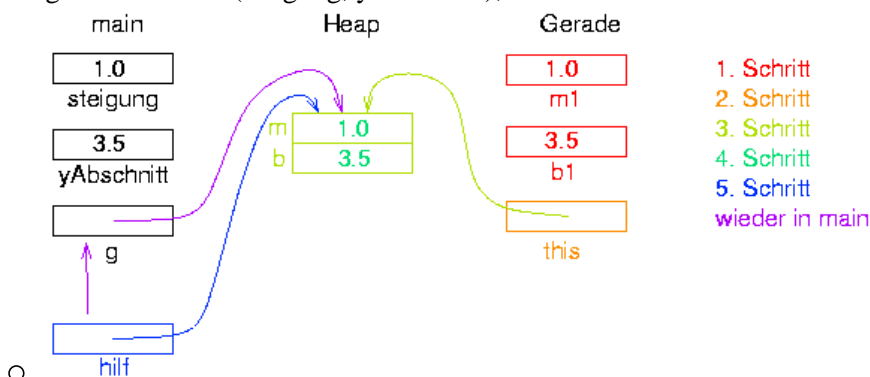
1. Parameterwerte werden kopiert (wie bei jedem Unterprogrammaufruf).
2. Zusätzliche Variable this vom Klassentyp (im Beispiel Gerade) wird im Stack des Konstruktors angelegt.
3. Im Heap wird Platz für das Objekt angelegt und Referenz darauf in this gespeichert.
4. Code des Konstruktors wird ausgeführt. Dabei kann die explizite Angabe der this-Referenz entfallen, d.h.
  - m ist identisch zu this.m
5. Referenz wird automatisch an den Aufrufer zurückgegeben mit (implizitem)
  - return this;

- Beispiel

```

double steigung = 1.0;
double yAbschnitt = 3.5;
Gerade g;
g = new Gerade(steigung, yAbschnitt);

```



- Spezielle Methode this:

- Aufruf eines bereits definierten Konstruktors innerhalb eines weiteren Konstruktors
- muss erste Anweisung des neuen Konstruktors sein
- ermöglicht, Standard-Initialisierungen in einem Konstruktor unterzubringen, den man in

allen anderen aufruft

○ Beispiel:

●

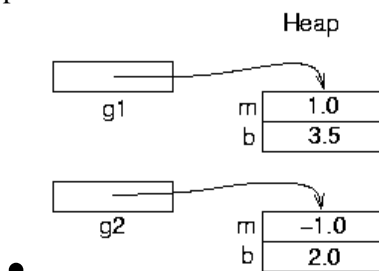
```
Gerade(double m1) {
 // erzeugt Gerade durch Ursprung
 this(m1, 0.0);
}
```

# Verwendung von Objekten

- Beispiel main in TestGerade.java:
  -

```
public static void main(String[] args) {
 Gerade g1 = new Gerade(1.0, 3.5);
 Gerade g2 = new Gerade(-1.0);
 g2.verschiebe(1.0, 1.0);
 System.out.println("Gerade 1: " + g1);
 System.out.println("Gerade 2: " + g2);
}
```

- Ausführung des Beispiels:
  - Speicherabbild am Ende



- - Ausgabe
    - Gerade 1:  $y = 1.0 \cdot x + 3.5$   
Gerade 2:  $y = -1.0 \cdot x + 2.0$
    - Ausgabe verwendet automatisch die toString()-Methode
      - $g1 \rightarrow g1.toString()$
    - da Verkettung (+) von String und anderem Typ (hier Gerade)
  - Methodenaufruf:
    - Methoden haben "unsichtbaren" zusätzlichen Parameter this
    - this enthält Referenz auf das Bezugsobjekt
    - im Beispiel
      - $verschiebe(double\ x, double\ y) \rightarrow verschiebe(double\ x, double\ y, Gerade\ this)$
      - $g2.verschiebe(1.0, 1.0) \rightarrow verschiebe(1.0, 1.0, g2)$
    - expliziter Bezug auf this kann innerhalb der Methoden weggelassen werden (wie im Konstruktor)
    - kein this-Parameter bei Konstruktoren und statischen Methoden (s.u.)
  - Aufgaben:
    - Aufgabe 7

# Zugriffskontrolle

- Zugriffsmöglichkeiten auf Komponenten:
  - öffentliche Datenfelder und Methoden
    - mit dem Attribut `public` gekennzeichnet
    - können von außen (von anderen Objekten aus) verwendet werden
  - private Datenfelder und Methoden
    - mit `private` bezeichnet
    - von außen unsichtbar
    - dienen für interne Zwecke der Klasse
  - kein Zugriffs-Attribut angegeben → Zugriff innerhalb desselben Pakets erlaubt
- Beispiel:
  -

```
public class Gerade {
 private double m; // Steigung
 private double b; // y-Achsenabschnitt

 public Gerade(double m, double b) { /* ... */ }
 public Gerade(double m) { /* ... */ }

 public void verschiebe(double x, double y) { /* ... */ }
 public String toString() { /* ... */ }

 private void verschiebeX(double x) { /* ... */ }
 private void verschiebeY(double y) { /* ... */ }
}
```

- öffentlich: Konstruktoren, `verschiebe`, `toString`
- privat: Datenfelder, `verschiebeX`, `verschiebeY`
- Schnittstelle einer Klasse:
  - alle öffentlich zugreifbaren Datenfelder und Methoden
  - legt die Klasse "nach außen" fest
  - erster und wichtigster Schritt beim Design einer Klasse
  - private Komponenten und alle Implementierungen können geändert werden, ohne dass ein Benutzer der Klasse sein Programm ändern muss
  - erlaubt z.B., Interna einer Klasse nachträglich zu ändern, um die Ausführungs-Geschwindigkeit zu erhöhen
  - Datenfelder meistens privat
    - Kapselung der Implementierung
    - erlaubt größere Flexibilität für spätere Änderungen
  - bei Bedarf Zugriffsmethoden, z.B.:

```
public double getSteigung() {
 return m;
}

public void setSteigung(double m1) {
 m = m1;
}
```

- Innere Klassen:

- Klassen, die innerhalb anderer Klassen definiert sind
- haben Zugriff auch auf die privaten Komponenten der umgebenden Klasse
- nützlich als Hilfsklassen, die nur zur Implementierung der umgebenden Klasse gebraucht werden
- häufig bei der Programmierung graphischer Oberflächen eingesetzt

# Klassenvariablen und -methoden

- Klassenvariablen:

- nicht mit einem einzelnen Objekt verknüpft, sondern mit der Klasse
- werden mit dem Schlüsselwort `static` deklariert
- Zugriff über den Klassennamen
- Beispiel:



```
public class Math {
 public static final double PI = 3.1415926;
 // ...
}

public class Schätzer {
 public double einfach(double daumen) {
 return Math.PI * daumen;
 }
}
```

- können z.B. auch benutzt werden, die Zahl der erzeugten Objekte festzuhalten

- Klassenmethoden:

- werden mit dem Schlüsselwort `static` deklariert
- Aufruf über den Klassennamen, z.B.
  - `double x = Math.sin(42.0);`
- benötigen kein Bezugsobjekt
- haben daher keinen impliziten `this`-Parameter
- können nur auf Klassenvariable zugreifen (da kein Objekt existiert)

- `main`-Methode:

- spezielle Klassenmethode
- deklariert als
  - `public static void main(String[] args)`
- Aufruf `java KLASSE` startet `KLASSE.main()`
- Parameter `args` enthält Kommandozeilenparameter, z.B.
  - `java TestKlasse bla foo 42`
  - `→ args = {"bla", "foo", "42"}`
- löst Henne-Ei-Problem
  - ohne Objekt keine Methoden
  - ohne Methode keine Erzeugung von Objekten

- Aufgaben:

- Aufgabe 8

# Vererbung

- Prinzip der Vererbung:
  - Erzeugung einer neuen Klasse durch Ableitung von einer vorhandenen Klasse (der Basisklasse)
  - die neue Klasse enthält alle Datenfelder und Methoden der Basisklasse, mit Ausnahme der Konstruktoren
- Abgeleitete Klasse:
  - kann zusätzliche Datenfelder und Methoden definieren (erweitert die Basisklasse)
  - kann Methoden der Basisklasse durch gleichnamige Methoden überschreiben
  - kann mit super.NAME auf überschriebene Datenfelder oder Methoden zugreifen
  - Objekt der abgeleiteten Klasse ist auch vom Typ der Basisklasse
- Beispiel:
  - Basisklasse Polygon, beschreibt allgemeines n-Eck

```
public class Polygon {
 private Punkt[] punkte; // Liste der Eckpunkte

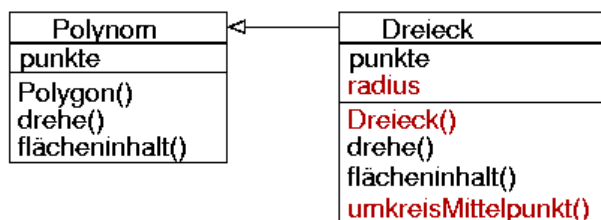
 public Polygon(Punkt[] p) { ... }
 public void drehe(double Winkel) { ... }
 public double flächeninhalt() { ... }
}
```

- abgeleitete Klasse Dreieck

```
public class Dreieck extends Polygon {
 private double radius;
 // zwischengespeichert wegen Performance

 public Dreieck(Punkt a, Punkt b, Punkt c) { ... }
 public Punkt umkreisMittelpunkt() { ... }
}
```

- bildlich:



- Dreieck "ist ein" Polygon in doppelter Hinsicht:
  1. enthält (bis auf den Konstruktor) alle Datenfelder und Methoden von Polygon
  2. kann überall dort verwendet werden, wo ein Polygon stehen soll, z.B.:

```
Polygon p; // nur eine Referenz!
Dreieck d = new Dreieck(..);
p = d;
// p zeigt auf ein Polygon, nämlich d
p.drehe(90.0);
```

- Dreieck ist ein besonderes Polygon, es kann mehr. Etwa mit den obigen Bezeichnungen:



```

Punkt mitte = d.umkreisMittelpunkt();

// !! Fehler:
Punkt mitte2 = p.umkreisMittelpunkt();
// geht nicht, denn ein Polygon hat keine
// Methode umkreisMittelpunkt - auch dann nicht,
// wenn p "in Wirklichkeit" ein Dreieck ist

```

- Konstruktoren und Vererbung:

- Kontruktor einer abgeleiteten Klasse

- kann Konstruktoren der Basisklasse zur Initialisierung der Basiskomponenten verwenden
- Aufruf mit super
- muss die erste Anweisung im Konstruktor sein

- Beispiel:

- 

```

public Dreieck(Punkt a, Punkt b, Punkt c) {
 super(new Punkt[] {a, b, c});
 // erzeugt Array aus a, b, c
}

```

- kein super-Aufruf verwendet →

- Compiler fügt automatisch super() ein
- meldet Fehler, wenn es keinen Konstruktor ohne Argumente (Standard-Konstruktor) gibt

- Zugriffskontrolle bei Vererbung:

- abgeleitete Klasse enthält zwar die privaten Datenfelder und Methoden der Basisklasse, kann sie aber nicht selbst verwenden
- weitere Zugriffsklasse protected
  - erlaubt den Zugriff nur für abgeleitete Klassen
  - "Familien-Angelegenheit" ("privat außer für meine Kinder")
  - Vorteil: leichtere Implementierung der Methoden für die abgeleiteten Klassen
  - Nachteil: abgeleitete Klassen müssen sich bei internen Änderungen der Basisklassen anpassen

- Beispiel Dreieck extends Polygon

- Problem

- punkte in Polygon privat
- → Dreieck kann umkreisMittelpunkt nicht implementieren

- Ausweg 1

- punkte wird protected
- schnelle Implementierung
- Dreieck sieht Interna von Polygon

- Ausweg 2

- Polygon hat Methode

```
public Punkt[] getPunkte()
```
- gut gekapselt
- ineffizienter für Dreieck

- Basisklasse Object:

- Java-Standardklasse, von der alle anderen Klassen abstammen
- Klassen ohne explizite Basisklasse direkt von Object abgeleitet

- stellt einige Grundfunktionen zur Verfügung, darunter:



| <b>Methode</b>           | <b>Bedeutung</b>                                   |
|--------------------------|----------------------------------------------------|
| String toString()        | gibt einen String aus Klassennamen und Code zurück |
| boolean equals(Object o) | prüft nur auf Gleichheit der Zeiger                |

- von allen Java-Standardklassen sinnvoll überschrieben
- sinnvoll auch für eigene Klassen

# Polymorphie

- Überschreiben von Methoden:

- eine in der Basisklasse definierte Methode wird auch in der abgeleiteten Klasse definiert

- Gründe:

- Kindklasse muss zusätzliche Daten berücksichtigen
- Kindklasse kann effizienter implementieren

- Beispiel:

- 

```
public class Polygon {
 // ..
 public double flächeninhalt() {
 //schwierige Flächenberechnung beim Polygon
 }
}

public class Dreieck extends Polygon {
 // ..
 public double flächeninhalt() {
 // einfache Flächenberechnung beim Dreieck
 }
}
```

- Auswahl der überschriebenen Methode:

- Beispiel

- 

```
class Dreiecktest {
 public static void main(String[] args) {
 Polygon[] p = new Polygon[2];
 p[0] = new Polygon();
 p[1] = new Dreieck();
 for (int i=0; i<2; i++) {
 System.out.println("Fläche: " +
 p[i].flächeninhalt());
 }
 }
}
```

- Welche Methode ist p[1].flächeninhalt:

- die von Polygon, denn p[1] ist ein Polygon,
- die von Dreieck, denn p[1] zeigt auf ein Dreieck?
- Antwort: die von Dreieck

- Regel:

- Zeigt eine Variable vom Basistyp auf ein Objekt vom Kindtyp, so wird bei überschriebenen Methoden die Version des Kinds verwendet. Dieses Verhalten nennt man Polymorphie.

- Kindmethode implementiert die gleiche Schnittstelle (hoffentlich mit gleicher Bedeutung) → es schadet nicht

- Kindmethode hat mehr Informationen → es kann sehr nützen

- Abstrakte Basisklasse:

- Klasse, die nicht alle ihre Methoden implementiert
- gekennzeichnet mit abstract, ebenso die Methoden

- Beispiel:



```
abstract public class Figur {
 // Komponenten und vollständige Methoden ...

 abstract double flächeninhalt();
}
```

- Zweck

- Basisklasse für Gruppe verwandter Klassen
- gemeinsame Methoden sind schon definiert
- spezielle Methoden mit gleicher Schnittstelle sind bekannt, aber noch nicht implementiert

- Interface:

- Spezialfall von abstrakter Basisklasse

- implementiert keine einzige Methode
- hat keine Variablen außer Konstanten (mit final bezeichnet)

- in Java eigener Datentyp

- abgeleitete Klassen

- müssen alle Methoden implementieren
- "erben" nichts (höchstens Konstanten)
- mit implements statt mit extends gekennzeichnet

- eine Klasse kann mehrere Interfaces implementieren

- Aufgaben:

- Aufgabe 9

# **Arbeiten mit Klassen-Bibliotheken**

- Pakete
- Grundlegende Klassen
- Fehlerbehandlung
- Ein- und Ausgabe

# Pakete

- Paket (package):
  - Menge von zusammengehörenden Klassen und Schnittstellen (Klassen-Bibliothek)
  - Paketname: meistens mehreren Komponenten, durch Punkt getrennt
  - Beispiele: java.lang, java.util.zip, javax.swing
  - Namen entsprechen Unterverzeichnissen, z.B.
    - java.lang → java\lang
    - java.util.zip → java\util\zip
  - Compiler sucht Paket-Verzeichnisse im aktuellen Verzeichnis oder mit der Umgebungsvariablen CLASSPATH
- Zugriff auf Klassen:
  - Klassen aus Paketen mit vorangestelltem Paketnamen, etwa
    - java.io.BufferedReader
    - = Klasse BufferedReader in Paket java.io
  - abkürzende Schreibweise mit Hilfe des import-Befehls vor der eigenen Klassendefinition

```
import java.io.*; // * = alle Klassen des Pakets

public class MeineKlasseMitEingabe {
 BufferedReader br;
 // etc...
}
```

- nützliche Neuerung in Java 1.5: statische Imports
  - statt
  - Math.sqrt(Math.sin(Math.PI/8.0))
  - einfach
  - import static java.lang.Math.\*;
  - sqrt(sin(PI/8.0))
- Standardklassen aus Paket java.lang direkt mit Klassennamen
- Erzeugen eigener Pakete:
  - Name des Pakets in package Anweisung
    - package mein.schönes.paket;
  - muss erste Anweisung einer Datei sein
  - keine package-Anweisung →
    - Teil des eigenen Standard-Pakets
    - ok für kleine Anwendungen und Tests

# Grundlegende Klassen

- Standard-Klassen von Java:
  - riesige Anzahl in vielen Paketen
  - in Dokumentation des JDK ausführlich beschrieben
  - besonders wichtige in Paket java.lang, u.a.
    - String
    - Math
    - System
    - Double, Integer, ...
- String:
  - Klasse zur Darstellung von konstanten Zeichenketten (variabel: StringBuffer)
  - syntaktische Besonderheiten
    - Bezeichnung von Strings mit Anführungszeichen "
    - Verkettung von Strings mit +
  - Methode toString zur Darstellung eines Objekts als String für alle Standardklassen definiert
  - wichtige Methoden

| Methoden von String                  | Bedeutung                        |
|--------------------------------------|----------------------------------|
| int length()                         | Anzahl der Zeichen im String     |
| char charAt(int index)               | Zeichen an der Position index    |
| String substring(int begin, int end) | Teilstring von begin bis end - 1 |
| int compareTo(String anotherString)  | lexikalischer Vergleich          |

- interne Darstellung von Strings ist verborgen
- Methoden zur Wandlung von String in Zahl in numerischen Klassen Double, Integer etc.
- Math:
  - Klasse, die die wichtigsten mathematischen Konstanten und Funktionen als statische Größen zusammenfasst
  - wichtige Felder und Methoden

| Felder und Methoden von Math | Bedeutung                                        |
|------------------------------|--------------------------------------------------|
| Math.PI, Math.E              | $\pi$ und e                                      |
| pow, exp, ln, sqrt           | Potenz, Exponentialfunktion, Logarithmus, Wurzel |
| cos, sin, tan, asin, ..      | trigonometrische Funktionen und Inverse          |
| min, max                     | Minimum, Maximum                                 |
| floor, ceil, round           | ganzzahlige Werte in der Nähe                    |
| random                       | Zufallszahl zwischen 0.0 und 1.0                 |

- System:
  - Klasse mit einigen betriebssystemnahen Funktionen
  - enthält die Zeiger auf Standard-Ein- und Ausgabe
  - wichtige Felder und Methoden

| <b>Felder und Methoden von System</b> | <b>Bedeutung</b>                                    |
|---------------------------------------|-----------------------------------------------------|
| System.in                             | Standard-Eingabe als InputStream                    |
| System.out, System.err                | Standard-Ausgabe- und -Fehler-Kanal als PrintStream |
| long currentTimeMillis()              | Zeit seit 1.1. 1970 in Millisekunden                |
| void exit(int)                        | beendet das laufende Programm                       |



# Fehlerbehandlung

- Grundproblem:
  - Programme haben Fehler oder treffen auf unerwartete Probleme
  - Beispiele:
    - Division durch 0
    - schreibgeschützte Datei
    - mangelnder Speicherplatz
  - Java erzeugt Ausnahmen, auf die im Programm reagiert werden kann
- Exception (Ausnahme):
  - Oberklasse aller Fehlertypen
  - zeigt Auftreten einer unerwarteten Situation an
  - genauer Typ (Unterklasse) gibt Aufschluss über Art des Problems
  - typische Beispiele:

| Klasse                    | Bedeutung                                              |
|---------------------------|--------------------------------------------------------|
| OutOfMemoryError          | kein Speicherplatz mehr bei einem new-Aufruf           |
| FileNotFoundException     | eine Datei des angegebenen Namens wurde nicht gefunden |
| ArithmeticException       | Integer-Division durch Null                            |
| IndexOutOfBoundsException | Array-Index ist zu groß oder negativ                   |

- Behandlung von Ausnahmen:
  - Syntax

```
try {
 // Anweisungen, die zu Ausnahmen führen können
} catch (ExceptionTyp1 e) {
 // Anweisungen zur Behandlung von ExceptionTyp1
} catch (ExceptionTyp2 e) {
 // Anweisungen zur Behandlung von ExceptionTyp2
}
```
  - Bedeutung
    - Ausnahme ausgelöst ⇒
    - umschließender try-Block wird gesucht
    - nachfolgender catch-Zweig des richtigen Typs wird angesprungen
    - Anweisungen des catch-Blocks werden ausgeführt
    - Programm macht ggf. hinter den catch-Blöcken weiter
  - Möglichkeiten im catch-Block
    - Fehler irgendwie beheben bzw. umgehen und weitermachen
    - aufräumen und aufhören
  - nützliche Methode in Exception: printStackTrace()
- Erzeugen von Ausnahmen:
  - Beispiel

```
void dangerous() throws Exception {
 // ...
 if (error condition) {
 throw new Exception("problems in f");
 }
 // ...
}
```

- Auslösen mit throw-Anweisung (throw new Exception())
- Anzeigen im Routinen-Kopf (throws Exception)
- Benutzer von dangerous muss
  - try-catch-Block zum Abfangen verwenden
  - oder empfangene Ausnahme nach oben durchreichen und dies mit throw anzeigen
- ggf. eigene Art von Ausnahme = von Exception abgeleitete Klasse

# Ein- und Ausgabe

- Paket java.io:
  - große Anzahl von Klassen
  - sehr flexibel
  - etwas unübersichtlich organisiert
  - hier nur die einfachsten Möglichkeiten
- Stream:
  - Strom von Daten, die nur seriell (nacheinander) bearbeitet werden können
  - InputStream = Daten treffen von einer Quelle ein
  - OutputStream = Daten werden an einen Empfänger gesendet
  - erlaubt einheitliche Verarbeitung ganz verschiedener Medien
  - Beispiele:
    - Tastatur/Bildschirm
    - Datei
    - String (Daten im Speicher)
    - Netzwerk-Verbindung
  - Verarbeitungsschema:
    - Stream öffnen
    - Daten lesen oder schreiben
    - Stream schließen
- Schreiben auf den Bildschirm (Textkonsole):
  - in Klasse System vordefiniertes Objekt out der Klasse PrintStream
  - automatisch geöffnet
  - Methoden print und println für alle einfachen Datentypen und String
- Lesen von der Tastatur:
  - in Klasse System vordefiniertes Objekt in der Klasse InputStream
  - automatisch geöffnet
  - kann nur nackte Bytes lesen mit read
- Filter-Klassen:
  - Klassen, die vorhandene Ein-/Ausgabe-Klassen um zusätzliche Funktionen erweitern
  - Beispiel InputStreamReader
    - wandelt nackte Bytes in (sprachenabhängige) Zeichen (char's)
  - Beispiel BufferedReader
    - Daten werden gepuffert und immer in ganzen Blöcken übertragen
    - Methode String readLine() holt ganze Zeile als String
    - BufferedWriter analog
- Arbeiten mit Dateien:
  - Grundtyp: FileReader oder FileWriter
  - Öffnen mit Konstruktor und Dateiname als Argument

```
FileReader meineDateiRoh
 = new FileReader("liesmich.txt");
```
  - in BufferedReader verwandeln (wesentlich schneller!)

```
BufferedReader meineDatei
 = new BufferedReader(meineDateiRoh);
```

- zeichenweise lesen oder schreiben mit read/write
- Beispielprogramm: CopyFile.java
- Aufgaben:
  - Aufgabe 10

# Programmieren graphischer Oberflächen

- Applets
- Dialogelemente
- Einfache Graphik
- Reagieren auf Ereignisse
- Anordnung der Komponenten

# Applets

- Applet:
  - in Java geschriebenes Programm ohne main-Methode
  - wird in eine HTML-Seite eingebettet
  - vom Browser beim Anzeigen der Seite automatisch ausgeführt
  - Problem:
    - alte Browser oft auf altem Java-Stand (vor 1.2)
    - Abhilfe: Java-Plugin von Sun
  - bei JDK mitgeliefertes Programm appletviewer zum Ansehen ohne Browser
- HTML-Grundmuster:
  - minimale Seite zum Anzeigen eines Applets
  -

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
 <HEAD>
 <TITLE>Applet-Test</TITLE>
 </HEAD>
 <BODY>
 <H1>Test von ZeichenApplet</H1>
 <APPLET CODE="ZeichenApplet.class" WIDTH="200" HEIGHT="160">
 </APPLET>
 </BODY>
</HTML>
```

- Parameter von Feld APPLET
  - CODE        Name der Basis-Klasse
  - WIDTH      Breite des Applet-Bereichs in HTML-Seite
  - HEIGHT     Höhe des Applet-Bereichs in HTML-Seite
- <APPLET> ist veraltet, <OBJECT> wird vom Internet Explorer völlig verdreht interpretiert, vgl. Hinweisseite des PhysBeans-Projekts
- Sicherheitsvorkehrungen bei Applets:
  - Applets aus dem Internet ladbar
  - dürfen eigenes System nicht beschädigen
  - grundsätzliche Einschränkungen für Applets:
    - kein Zugriff auf lokale Dateien
    - kein Start lokaler Programme
    - kein Zugriff auf andere Rechner im Netzwerk
    - Fenster außerhalb des Browser-Fensters nur mit Warnung
- Klasse JApplet:
  - Basisklasse für eigene Applets
  - enthalten im Paket javax.swing
  - enthält Methoden, die vom Browser aufgerufen werden:

Methoden von Applet	Bedeutung
void init()	beim Laden in den Browser
void start()	beim Anzeigen der HTML-Seite des Applets
void stop()	beim Anzeigen einer anderen Seite

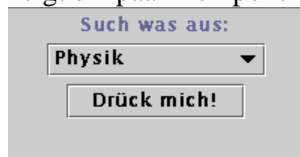
- sind in JApplet sinnvoll vordefiniert
- werden zumindest teilweise in eigener, von JApplet abgeleiteter Klasse überschrieben
- hat verschiedene Bereiche (Panee) für Menüs und Benutzerkomponenten (Schaltflächen, Auswahlboxen etc.)
- i.f. nur wichtig das ContentPane
  - abgeleitet von java.awt.Container
  - zugreifbar durch

```
Container pane = getContentPane();
```
  - Träger der Benutzerkomponenten
  - hinzufügen von Komponenten mit

```
pane.add(KOMPONENTE);
```
- Schema eines einfachen Applets:
  - von JApplet ableiten
  - in init zusammensetzen
  - dazu ContentPane besorgen und mit add Komponenten hinzufügen

# Dialogelemente

- Komponenten:
  - graphische Bedienungselemente
  - große Auswahl im Paket javax.swing enthalten
  - Beispiele:
    - Schaltknopf
    - Hinweistext
    - Auswahlbox
  - werden mit add in ein ContentPane eingefügt
- Schaltknopf:
  - Klasse JButton
  - Konstruktor mit Beschriftungstext
  - zusätzlich auch ein Icon möglich
- Hinweistext:
  - Klasse JLabel
  - Konstruktor mit Text
- Auswahlbox:
  - Klasse JComboBox
  - Konstruktor mit Liste von Auswahltexten
- Beispielprogramm HalloApplet
  - zeigt ein paar Komponenten in einem Applet



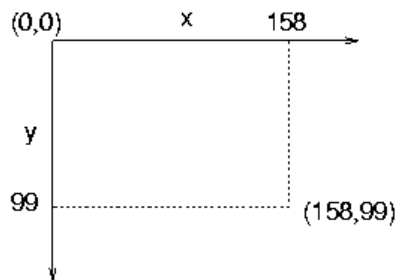
- Source
- Applet
- setLayout sorgt für schönere Anordnung (genauer s.u.)



# Einfache Graphik

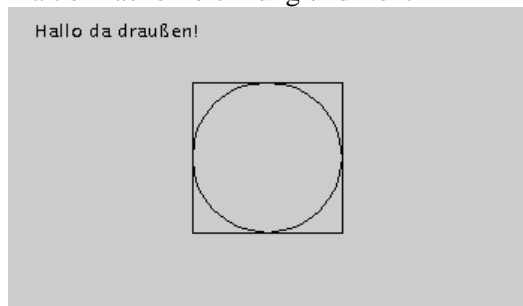
- JPanel:
  - Komponente zum Zusammenfassen anderer Komponenten
  - kann auch als Basis zum Zeichnen benutzt werden
  - hat Methode `paintComponent(Graphics g)`, die nur den Hintergrund löscht
  - Nutzungsweise:
    - leite eigene Klasse von JPanel ab
    - überschreibe darin `paintComponent`
    - bringe dort die eigentlichen Zeichenroutinen unter
  - hat Methoden `int getWidth()` und `int getHeight()` zur Bestimmung der Größe der Zeichenfläche

- Graphics:
  - Klasse mit grundlegenden Zeichenroutinen
  - legt Malfarben, Fonts etc. fest
  - definiert Koordinatensystem:



- Argument von `paintComponent`
- einige Methoden
  - `void drawLine(int x1, int y1, int x2, int y2)`  
malt gerade Linie von  $(x1, y1)$  nach  $(x2, y2)$
  - `void drawRect(int x, int y, int breite, int hoehe)`  
zeichnet Rechteck der Größe `breite x hoehe`  
linke obere Ecke ist  $(x, y)$
  - `void fillRect(int x, int y, int breite, int hoehe)`  
wie `drawRect`, aber ausgefülltes Rechteck
  - `void drawOval(int x, int y, int breite, int hoehe)`  
zeichnet ein Oval (ggf. Kreis), das in ein Rechteck  
`breite x hoehe` am Punkt  $(x, y)$  passt
  - `void drawString(String text, int x, int y)`  
zeichnet Text an Stelle  $(x, y)$
- Color:
  - Klasse zur Festlegung von Farben
  - im Paket `java.awt` enthalten
  - Konstruktor mit Werten für rot, grün, blau zwischen 0 und 255
  - Abfrage mit `getRed`, `getGreen`, `getBlue`
  - einige vordefinierte Farben wie `Color.black`, `Color.red` etc.
  - Farbe im Graphics-Objekt festlegen, etwa mit
    - `g.setColor(Color.green);`
- Beispielprogramm `ZeichenApplet.java`

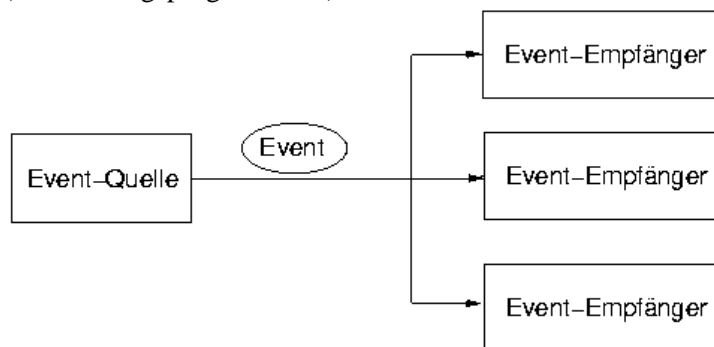
- malt einfache Zeichnung und Text



- Source
- Applet
- Bemerkungen
  - Applet hat nur eine Komponente, das ZeichenBrett
  - ZeichenBrett ist ein JPanel, es überschreibt paintComponent
  - Text erscheint an fester Stelle
  - Quadrat und Rechteck werden in die Mitte des Fensters gemalt
- Aufgaben:
  - Aufgabe 11

# Reagieren auf Ereignisse

- Arten von Ereignissen:
  - nach Verursacher
    - Benutzer direkt, z.B. Maus bewegt
    - Benutzer indirekt, z.B. verdecktes Fenster wird sichtbar
    - Programm selbst, z.B. Wartezeit abgelaufen
  - nach Bedeutung
    - "physikalisch" wie Mausbewegung, Tastatureingabe
    - "logisch" wie Menüauswahl, Texteingabe
  - als Klassen dargestellt
    - abgeleitet von der Klasse EventObject
    - meistens im Paket java.awt.event
    - z.B. MouseEvent, KeyEvent, ActionEvent
- Delegationsmodell:
  - Trennung von Sender (z.B. Benutzeroberfläche) und Empfänger (Anwendungsprogrammteil)



- Event-Quelle (Source)
    - z.B. graphische Komponente, die "betätigt" wird
    - erzeugt entsprechende Art von EventObject, z.B. ActionEvent
    - schickt es an alle interessierten Empfänger
  - Event
    - enthält Information über Sender
    - häufig Zusatzinformationen (welcher Menüeintrag, wo geklickt etc.)
  - Event-Empfänger (Listener)
    - hat sich bei der Quelle angemeldet
    - implementiert eine geeignete Schnittstelle, z.B. ActionListener
    - stellt dadurch vorgegebene Funktionen zur Verfügung, z.B. actionPerformed()
- Beispielablauf:
    - MeinListener hat sich beim Button angemeldet
    - Benutzer klickt Button an
    - Button erzeugt ActionEvent e
    - Button schickt Event zum Listener
      - konkret: es ruft Methode actionPerformed(e) von MeinListener auf
    - MeinListener reagiert auf das Event
      - konkret: MeinListener.actionPerformed(e) wird ausgeführt
  - Praktische Programmierung:

- beim Zusammensetzen der graphischen Oberfläche (im init()) Listener anmelden



```

 JButton button = new JButton("Drück mich!");
 button.addActionListener(new MyButtonListener());

```

- eigene Klasse MyButtonListener als ActionListener definieren



```

 class MyButtonListener implements ActionListener { }

```

- Listener häufig als innere Klasse

- → einfacher Zugriff auf die Elemente der Oberfläche

- Methode actionPerformed mit gewünschter Reaktion implementieren



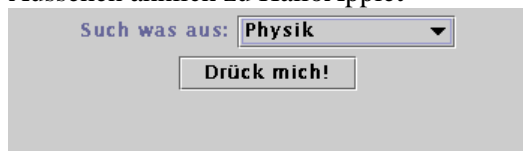
```

 public void actionPerformed(ActionEvent e) {
 // tauscht Liste der ComboBox aus
 switchList();
 }

```

- Beispielprogramm EventApplet:

- Aussehen ähnlich zu HalloApplet



- Verhalten

- Klick auf Button → Auswahlliste wechselt
- Auswahl eines Elements → Antwort als Textlabel

- Source

- Applet

- Erläuterungen zum Programm:

- Deklaration von title etc. als Datenfelder der Klasse

- nicht mehr als lokale Variable im init
- erlaubt Zugriff durch Listener

- ActionEvents werden erzeugt bei

- JButton wurde angeklickt
- in JComboBox wurde ein Element ausgewählt

- Methode von JLabel

- void setText(String s)  
Text des Labels wird durch s ersetzt

- Methoden von JComboBox

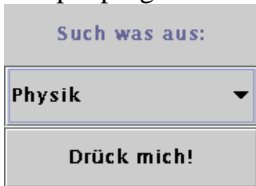
- int getSelectedIndex()  
gibt Nummer des angeklickten Elements zurück
- void removeAllItems()  
löscht alle Einträge in der Liste
- void addItem(String s)  
fügt ein Element zur Liste hinzu

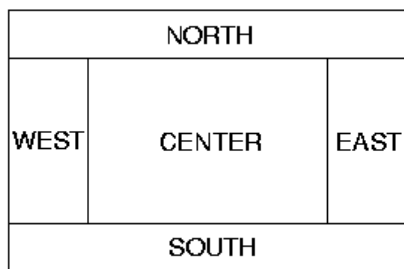
- Implementierung von MyButtonListener als Methode switchList()

- erhöht Übersichtlichkeit durch kurze Listener-Klasse

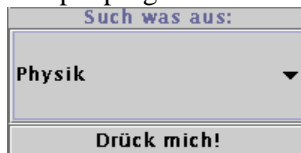
- erlaubt Aufruf von switchList ohne Event (z.B. zum Testen)

# Anordnung der Komponenten

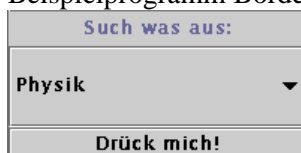
- Layout-Manager:
  - bestimmen Anordnung der Komponenten
    - Grundaufteilung je nach Typ des Layout-Managers
    - berücksichtigen z.T. "Wunsch-Größen" der einzelnen Komponenten
    - zwingen z.T. Komponenten bestimmte Größen auf
  - diverse Arten, u.a.
    - FlowLayout
    - GridLayout
    - BorderLayout
  - Festlegen des LayoutManager-Typs
    - als Argument im Konstruktor
    - `JPanel upperPanel = new JPanel(new BorderLayout());`
    - nachträglich mit `setLayout`
    - `upperPanel.setLayout(new FlowLayout());`
- FlowLayout:
  - Layout-Politik
    - ordnet alle Komponenten hintereinander an
    - belässt den Komponenten ihre natürliche Größe
    - Zeile voll → weitere Komponenten in neuer Zeile
    - alle Zeilen zentriert
  - Default-Layout des JPanel
- GridLayout:
  - Layout-Politik
    - erzeugt rechteckiges Gitter aus  $n \times m$  Zellen
    - alle Zellen gleich groß
    - ordnet Komponenten der Reihe nach in das Gitter ein
    - zwingt Komponenten Zellengröße auf
  - Anzahl der Zellen aus Konstruktor
    - `new GridLayout(3, 2)`
  - Beispielprogramm GridLayoutApplet
    - Sourcen
    - Applet
  - Text eines JLabel zentriert mit
    - `title = new JLabel("Such was aus:", JLabel.CENTER);`
- BorderLayout:
  - Layout-Politik teilt Fläche in fünf Teilbereiche



- NORTH und SOUTH: Komponenten haben natürliche Höhe und maximale Breite
- WEST und EAST: Komponenten haben natürliche Breite und maximale Höhe
- CENTER: Komponente hat maximale Größe
- Default-Layout des ContentPane
- Angabe des Bereiches als Argument beim add
  - `p.add(title, BorderLayout.NORTH);`
- Beispielprogramm BorderLayoutApplet

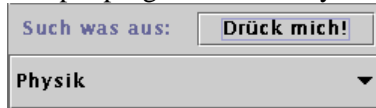


- Sourcen
- Applet
- Ränder:
  - Standard abhängig vom Komponententyp
    - JButton hat Rand um Text
    - JLabel hat keinen Rand
  - nachträgliches Hinzufügen von verschiedensten Rändern möglich, z.B.
    - einfacher Platz
    - Umrandungen
    - Beschriftungen
    - Girlanden
  - Ränder implementieren Interface `javax.swing.border.Border`
    - Border weiss, wie man einen Rand malt
    - ein Objekt für alle gleichartigen Ränder aller Komponenten
  - Rand wird erzeugt mit der Border-Fabrik
    - `Border simpleSpace = BorderFactory.createEmptyBorder(5,5,5,5);`
  - Komponente bekommt Rand mit `setBorder`
    - `title.setBorder(simpleSpace);`
  - Beispielprogramm BorderLayoutWithBorderApplet



- Sourcen
- Applet
- Kompliziertere Layouts:
  - mehrere Komponenten in einem JPanel gruppieren
  - jedes JPanel mit eigenem, geeigneten Layout

- Ränder für jedes Panel und für jede Komponente nach Bedarf
- vor der Programmierung Skizze mit gewünschter Anordnung!
- Beispielprogramm NiceLayoutApplet



- Sourcen
- Applet
- Aufbau der Oberfläche
  - Text und Button in eigenes JPanel upperPanel
  - upperPanel und contentPane haben BorderLayout
  - upperPanel in NORTH von contentPane → Höhe passt sich an Komponenten an
  - JComboBox in CENTER → nimmt gesamten restlichen Raum ein
- Aufgaben:
  - Aufgabe 12
  - Aufgabe 13



# Aufgaben

- Aufgabe 1
- Aufgabe 2
- Aufgabe 3
- Aufgabe 4
- Aufgabe 5
- Aufgabe 5a
- Aufgabe 6
- Aufgabe 7
- Aufgabe 8
- Aufgabe 8a
- Aufgabe 9
- Aufgabe 10
- Aufgabe 11
- Aufgabe 12
- Aufgabe 13

# Aufgabe 1

- Erzeugen Sie die Datei Primzahl.java für das Beispielprogramm
  1. Übersetzen Sie das Programm und führen Sie es mit verschiedenen Eingaben aus.
  2. Erweitern Sie das Programm, so dass es auch auf Eingabe negativer Zahlen sowie bei 1 und 2 richtig reagiert.
- Hinweise:
  - Programmtext erzeugen durch
    - Abtippen
    - mit Copy und Paste aus dem Browser in einen Editor kopieren
    - Datei im Browser abspeichern und alles HTML-Beiwerk löschen
  - Nicht vergessen: Die Klasse Primzahl **muss** in einer Datei mit dem Namen Primzahl.java stehen.
  - Verknüpfung von Bedingungen durch logisches UND in Java mit &&
    -

```
if (BEDINGUNG1 && BEDINGUNG2) {
 ANWEISUNGEN1
} else {
 ANWEISUNGEN2
}
```

- lauffähiges Applet
- Source

# Primzahl.java

```
import java.io.*;

public class Primzahl {

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

 BufferedReader in = new BufferedReader(
 new InputStreamReader(System.in));

 // Zahl erfragen
 String s; // String für eingegebene Zeile
 System.out.println("Positive ganze Zahl eingeben:");
 s = in.readLine();
 int zahl = Integer.parseInt(s);

 boolean isPrim = true; // bis zum Beweis des Gegenteils
 int teiler = 1; // gefundener Teiler

 // Eingabe ok
 if (zahl <= 0) {
 System.out.println("ungültige Eingabe: nicht positiv!");
 System.exit(-1);
 }

 if (zahl == 1) { // Spezialfall 1
 isPrim = false;
 } else if ((zahl % 2 == 0) && (zahl != 2)) { // durch 2 teilbar
 isPrim = false;
 teiler = 2;
 } else {
 int max = (int) Math.floor(Math.sqrt((double) zahl));
 for (int i=3; i<=max; i=i+2) {
 if (zahl % i == 0) { // durch ungerade Zahl kleiner Wurzel teilbar
 isPrim = false;
 teiler = i;
 }
 }
 }

 // Ausgabe des Ergebnisses
 if (isPrim) {
 System.out.println(zahl + " ist eine Primzahl");
 } else {
 System.out.println(zahl + " ist durch " + teiler + " teilbar");
 }
 }
}
```

## Aufgabe 2

- Schreiben Sie ein Programm, das eine Jahreszahl einliest und ausgibt, ob das Jahr ein Schaltjahr ist.
- Hinweise:
  - j ist Schaltjahr  $\Leftrightarrow$ 
    - j ist durch 4 teilbar UND
    - j ist nicht durch 100 teilbar ODER j ist durch 400 teilbar
  - logische Verknüpfungen UND bzw. ODER zwischen Bedingungen werden in Java mit `&&` bzw. `||` ausgedrückt
- lauffähiges Applet
- Source

# Schaltjahr.java

```
import java.io.*;

public class Schaltjahr {

 public static void main(String[] args) throws IOException {
 // prüft auf Schaltjahr

 BufferedReader in = new BufferedReader(
 new InputStreamReader(System.in));

 // Jahr erfragen
 String s; // String für eingegebene Zeile
 System.out.println("Jahreszahl eingeben:");
 s = in.readLine();
 int jahr = Integer.parseInt(s);

 if ((jahr % 4 == 0) &&
 ((jahr % 100 != 0) || (jahr % 400 == 0))) {
 System.out.println("Das Jahr " + jahr + " ist ein Schaltjahr");
 } else {
 System.out.println("Das Jahr " + jahr + " ist kein Schaltjahr");
 }
 }
}
```

## Aufgabe 3

- Schreiben Sie ein Programm, das Zähler und Nenner eines Bruchs einliest und ihren größten gemeinsamen Teiler (GGT) sowie den gekürzten Bruch ausgibt.

- Hinweise

- Am einfachsten berechnet man den GGT mit dem Euklidischen Algorithmus

- 

```
wiederhole, bis rest = 0
 rest = zaehler % nenner
 zaehler = nenner
 nenner = rest
GGT = zaehler
```

- Solche Schleifen werden in Java mit der while-Konstruktionen erzeugt

- 

```
while (BEDINGUNG) {
 tu was;
}
```

- lauffähiges Applet
- Source

# Kuerzen.java

```
import java.io.*;

public class Kuerzen {

 public static void main(String[] args) throws IOException {
 // kürzt einen Bruch

 BufferedReader in = new BufferedReader(
 new InputStreamReader(System.in));

 System.out.println("Zähler eingeben: ");
 int zaehler = Integer.parseInt(in.readLine());
 System.out.println("Nenner eingeben: ");
 int nenner = Integer.parseInt(in.readLine());

 // berechnet ggt nach dem Euklidischen Algorithmus
 int x = zaehler;
 int y = nenner;
 int rest = 0;
 while (y != 0) {
 rest = x % y;
 x = y;
 y = rest;
 }
 int ggt = x;

 System.out.println("größter gemeinsamer Teiler: " + ggt);
 System.out.println("gekürzter Bruch: " + zaehler/ggt + "/" + nenner/ggt);
 }
}
```

## Aufgabe 4

- Schreiben Sie ein Programm zur Umrechnung von Fahrenheit in Celsius gemäß
  - $T/^{\circ}\text{F} = 9/5 T/^{\circ}\text{C} + 32$
- Der Benutzer gebe dazu den kleinsten und größten Fahrenheit-Wert an. Das Programm erstellt dann eine Umrechnungstabelle in Schritten von  $1^{\circ}\text{F}$ . Verwenden Sie ein Array für die Celsius-Werte und trennen Sie Berechnung und Ausgabe.
- Ergänzung - Machen Sie das Programm narrensicher:
  - Es soll mit nicht-ganzzahligen Grenzen für die Fahrenheit-Tabelle umgehen können.
  - Es soll auch funktionieren, wenn man versehentlich erst den maximalen, dann den minimalen Wert eingibt.
- lauffähiges Applet
- Source



# Fahrenheit.java

```
import java.io.*;

public class Fahrenheit {

 public static void main(String[] args) throws IOException {
 // berechnet Tabelle Fahrenheit - Celsius

 BufferedReader in = new BufferedReader(
 new InputStreamReader(System.in));

 // Grenzen erfragen
 String s; // String für eingegebene Zeile
 System.out.println("Kleinster gewünschter Fahrenheit-Wert:");
 s = in.readLine();
 double min = Double.parseDouble(s);
 System.out.println("Größter gewünschter Fahrenheit-Wert:");
 s = in.readLine();
 double max = Double.parseDouble(s);

 // falls verkehrt herum: min und max vertauschen
 if (min > max) {
 double t = max;
 max = min;
 min = t;
 }

 // Array für Werte erzeugen
 int anzahl = (int) Math.round(max - min) + 1;
 double[] celsius = new double[anzahl];

 // Werte berechnen
 for (int i=0; i < anzahl; i++) {
 double fahrenheit = min + i;
 celsius[i] = 5.0/9.0*(fahrenheit - 32);
 }

 // Ausgabe des Ergebnisses
 System.out.println("Fahrenheit Celsius");
 for (int i = 0; i < anzahl; i++) {
 double fahrenheit = min + i;
 System.out.println(fahrenheit + " " + celsius[i]);
 }
 }
}
```

## Aufgabe 5

- Schreiben Sie ein Programm, das das Pascalsche Dreieck bis zu einer vorgegebenen Zeile berechnet und ausgibt.
- Hinweise:
  - Im Pascalschen Dreieck ist jede Zahl Summe der beiden über ihr stehenden, der Rand besteht aus Einsen.
  - Verwenden Sie ein normales 2d-Array für die Elemente des Pascalschen Dreiecks, bei dem die rechte obere Hälfte leer bleibt. Wenn Sie den Speicherplatz einsparen wollen, können Sie auch ein dreieckiges Array erzeugen, indem Sie verschieden lange Zeilen verwenden.
- lauffähiges Applet
- Source

# Pascal.java

```
import java.io.*;

public class Pascal {

 public static void main(String[] args) throws IOException {
 // berechnet das Pascalsche Dreieck

 BufferedReader in = new BufferedReader(
 new InputStreamReader(System.in));

 int[][] pascal;

 // Anzahl der Zeilen erfragen
 String s; // String für eingegebene Zeile
 System.out.println("Anzahl der Zeilen eingeben:");
 s = in.readLine();
 int nMax = Integer.parseInt(s) - 1; // beginnt bei n = 0

 // Berechnen der Werte
 pascal = new int[nMax + 1][]; // Platz für die Spalten
 for (int n=0; n <= nMax; n++) {
 pascal[n] = new int[n+1]; // Platz für eine Zeile
 pascal[n][0] = 1;
 for (int k=1; k < n; k++) {
 pascal[n][k] = pascal[n-1][k-1] + pascal[n-1][k];
 }
 pascal[n][n] = 1;
 }

 // Ausgabe des Ergebnisses
 for (int n=0; n <= nMax; n++) {
 for (int k=0; k <= n; k++) {
 System.out.print(pascal[n][k] + " ");
 }
 System.out.println();
 }
 }
}
```

## Aufgabe 5a

- Schreiben Sie ein Programm, das die Primfaktoren einer eingegeben Zahl ausgibt.
- Hinweise:
  - Wie bei der Primzahl-Bestimmung wird zuerst ein Teiler T bestimmt (2 oder eine ungerade Zahl kleiner als die Wurzel). Die ursprüngliche Zahl wird durch T geteilt und das Verfahren wiederholt. Ist man bei 1 angekommen, hat man alle Primfaktoren gefunden.
- lauffähiges Applet
- Source

# Faktorzerlegung.java

```
import java.io.*;

public class Faktorzerlegung {

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

 BufferedReader in = new BufferedReader(
 new InputStreamReader(System.in));

 // Zahl erfragen
 String s; // String für eingegebene Zeile
 System.out.println("ganze Zahl größer 1 eingeben:");
 s = in.readLine();
 int zahl = Integer.parseInt(s);

 // Eingabe ok
 if (zahl <= 1) {
 System.out.println("ungültige Eingabe!");
 System.exit(-1);
 }

 while (zahl > 1) {
 // Teiler finden und abspalten
 if (zahl % 2 == 0) { // durch 2 teilbar
 System.out.print("2 ");
 zahl /= 2;
 } else {
 boolean isPrim = true;
 int max = (int) Math.floor(Math.sqrt((double) zahl));
 for (int i=3; i<=max; i+=2) {
 if (zahl % i == 0) { // durch ungerade Zahl kleiner Wurzel teilbar
 System.out.print(i + " ");
 zahl /= i;
 isPrim = false;
 break; // for-Schleife verlassen
 }
 }
 if (isPrim) {
 System.out.println(zahl);
 zahl = 1;
 }
 }
 }
 }
}
```

## Aufgabe 6

- Schreiben Sie eine Routine `static void sort(double[] a)`, die ein gegebenes Array von Zahlen `a` in aufsteigender Reihenfolge sortiert. Testen Sie Ihre Routine mit einem Hauptprogramm, das ein Array von Zufallszahlen erzeugt und die Zeit für das Sortieren misst.
- Wettbewerb:
  - Wessen Routine sortiert am schnellsten? Gemessen wird, wieviele Zahlen die jeweiligen Sort-Routinen auf meinem Rechner in meinem Hauptprogramm in einer Minute sortieren können.
  - Achtung: Die Sieger müssen ihr Verfahren erklären können!
- Hinweise:
  - Überlegen Sie sich ein Verfahren zum Sortieren. Sie werden vermutlich eine Routine
    - `vertausche(array, i, j)`
  - gut gebrauchen können, die die Werte `a[i]` und `a[j]` des Arrays vertauscht.
  - Zufallszahlen zwischen 0 und 1 erhalten Sie mit
    - `double Math.random()`
  - Die aktuelle Zeit in Millisekunden seit dem 1.1.1970 liefert
    - `long System.currentTimeMillis()`
  - Zum Starten mit sehr großen Arrays reservierten Speicherplatz (Heap) vergrößern mit
    - `java -Xmx100m MeinSort`
- Beispielhaft seien hier zwei verschiedene Lösungen gezeigt:
  - Sortieren mit dem einfachen Bubblesort
    - lauffähiges Applet
    - Source
  - Sortieren mit dem sehr schnellen Quicksort
    - lauffähiges Applet
    - Source
  - genauere Erklärungen zur Funktionsweise im Kurs Programmiermethoden und Algorithmen

# BubbleSort.java

```
import java.io.*;

public class BubbleSort {
 // Beispielprogramm für Bubblesort-Algorithmus

 public static void main(String[] args) throws IOException {
 // erzeugt ein Array von Zufallszahlen und gibt es sortiert aus

 BufferedReader in = new BufferedReader(
 new InputStreamReader(System.in));

 // Größe des Arrays erfragen
 String s; // String für eingegebene Zeile
 System.out.println("Anzahl der zu sortierenden Werte eingeben:");
 s = in.readLine();
 int anzahl = Integer.parseInt(s);
 // abfragen, ob Liste ausgegeben werden soll
 System.out.println("Soll die sortierte Liste ausgegeben werden (j/n):");
 s = in.readLine();
 boolean mitAusgabe = false;
 if (s.compareTo("j") == 0) {
 mitAusgabe = true;
 }

 // erzeuge die Liste
 double[] liste = new double[anzahl];

 // fülle die Liste mit Zufallszahlen
 for (int i = 0; i < anzahl; i++) {
 liste[i] = Math.random();
 }

 // sortiere die Liste und miss die benötigte Zeit
 long timeBefore = System.currentTimeMillis();
 sort(liste);
 long timeAfter = System.currentTimeMillis();
 double timeSpent = (timeAfter - timeBefore)/1000.0; // in s
 System.out.println("Zeit zum Sortieren: " + timeSpent + "s");

 // gib die Liste aus, falls gewünscht
 if (mitAusgabe) {
 for (int i = 0; i < anzahl; i++) {
 System.out.println(liste[i]);
 }
 }
 }

 // Sortier-Routine nach dem Bubblesort-Algorithmus
 static void sort(double[] a) {
 boolean nochmal = true;

 while (nochmal) {
 // laufe solange durch das Array, bis keine Vertauschungen mehr nötig waren
 nochmal = false;
 for (int i = 0; i < a.length - 1; i++) {
 if (a[i] > a[i + 1]) {
 vertausche(a, i, i + 1);
 nochmal = true;
 }
 }
 }
 }
}
```

```
 }
 }
}

// vertausche a[i] und a[j]
static void vertausche(double[] a, int i, int j) {
 double temp = a[i];
 a[i] = a[j];
 a[j] = temp;
}
}
```



# QuickSort.java

```
import java.io.*;

public class QuickSort {
 // Beispielprogramm für Quicksort-Algorithmus

 public static void main(String[] args) throws IOException {
 // erzeugt ein Array von Zufallszahlen und gibt es sortiert aus

 BufferedReader in = new BufferedReader(
 new InputStreamReader(System.in));

 // Größe des Arrays erfragen
 String s; // String für eingegebene Zeile
 System.out.println("Anzahl der zu sortierenden Werte eingeben:");
 s = in.readLine();
 int anzahl = Integer.parseInt(s);
 // abfragen, ob Liste ausgegeben werden soll
 System.out.println("Soll die sortierte Liste ausgegeben werden (j/n):");
 s = in.readLine();
 boolean mitAusgabe = false;
 if (s.compareTo("j") == 0) {
 mitAusgabe = true;
 }

 // erzeuge die Liste
 double[] liste = new double[anzahl];

 // fülle die Liste mit Zufallszahlen
 for (int i = 0; i < anzahl; i++) {
 liste[i] = Math.random();
 }

 // sortiere die Liste und miss die benötigte Zeit
 long timeBefore = System.currentTimeMillis();
 sort(liste);
 long timeAfter = System.currentTimeMillis();
 double timeSpent = (timeAfter - timeBefore)/1000.0; // in s
 System.out.println("Zeit zum Sortieren: " + timeSpent + "s");

 // gib die Liste aus, falls gewünscht
 if (mitAusgabe) {
 for (int i = 0; i < anzahl; i++) {
 System.out.println(liste[i]);
 }
 }
 }

 static void sort(double[] liste) {
 quicksort(liste, 0, liste.length - 1);
 }

 static void quicksort(double[] a, int oben, int unten) {
 int high = oben;
 int low = unten;
 double x = a[(high+low)/2]; // "mittleres" Element

 do {
```

```

while (a[high] < x) { // suche großes Element bei den Kleinen
 high++;
}
while (a[low] > x) { // suche kleines Element bei den Großen
 low--;
}
if (high <= low) {
 vertausche(a, high, low);
 high++;
 low--;
}
} while (high <= low);

if (oben < low) { // Abbruchbedingung
 quicksort(a, oben, low);
}

if (high < unten) { // Abbruchbedingung
 quicksort(a, high, unten);
}
}

// vertausche a[i] und a[j]
static void vertausche(double[] a, int i, int j) {
 double temp = a[i];
 a[i] = a[j];
 a[j] = temp;
}
}

```

## Aufgabe 7

- Erweitern Sie die Beispielklasse Gerade:
  - Ändern Sie toString so, dass es auch bei negativem b richtig funktioniert.
  - Schreiben Sie eine Methode drehe, die die Gerade um einen gegebenen Winkel dreht.
  - Schreiben Sie eine Methode, die den Schnittpunkt der betrachteten mit einer gegebenen Geraden zurückgibt.
- Hinweis:
  - Um einen Punkt zurückgeben zu können, definieren Sie am besten eine einfache Klasse Punkt.
- Testen Sie alles mit einer geeigneten main-Methode.
- lauffähiges Applet
- Sourcen:
  - Punkt.java
  - Gerade.java
  - TestGerade.java

# Punkt.java

```
public class Punkt {
 // einfache Klasse für 2d-Punkt

 private double x;
 private double y;

 public Punkt(double x1, double y1) {
 x = x1;
 y = y1;
 }

 public String toString() {
 // Ausgabe: (x, y)
 return "(" + x + ", " + y + ")";
 }
}
```

# Gerade.java

```
public class Gerade {
 // beschreibt Gerade in zwei Dimensionen

 // Datenfelder
 private double m; // Steigung
 private double b; // y-Achsenabschnitt

 // Methoden
 // Konstruktor
 public Gerade(double m1, double b1) {
 m = m1;
 b = b1;
 }

 public void verschiebe(double x, double y) {
 // verschiebt Gerade um Vektor (x, y)
 verschiebeX(x);
 verschiebeY(y);
 }

 public String toString() {
 // Ausgabe: y = m*x + b
 // an Vorzeichen von b anpassen
 if (b > 0) {
 return "y = " + m + "*x + " + b;
 } else if (b < 0) {
 return "y = " + m + "*x - " + (-b);
 } else {
 // b == 0!
 return "y = " + m + "*x";
 }
 }

 private void verschiebeX(double x) {
 // verschiebt Gerade um x in x-Richtung
 b = b - m*x;
 }

 private void verschiebeY(double y) {
 // verschiebt Gerade um y in y-Richtung
 b = b + y;
 }

 public void drehe(double alpha) {
 // drehe Gerade um Winkel alpha (im Bogenmaß) um den Nullpunkt

 // neue Steigung aus Addition der Winkel
 double beta = Math.atan(m); // Winkel der Geraden
 m = Math.tan(alpha + beta);

 // neuer y-Achsenabschnitt aus Punkt-Steigungsform
 // alter Punkt P = (0,b)
 // nach Drehung: P' = (-b sin(alpha), b cos(alpha))
 // b' = Py' - m'*Px'
 b = b*(Math.cos(alpha) + m*Math.sin(alpha));
 }

 public Punkt berechneSchnittpunkt(Gerade g1) {
```

```
// bestimmt den Schnittpunkt der aktuellen Geraden und der Geraden g1

double xS = - (b - g1.b)/(m - g1.m);
double yS = m*xS + b;

Punkt schnittpunkt = new Punkt(xS, yS);
return schnittpunkt;
}
}
```

# TestGerade.java

```
import java.io.*;

public class TestGerade {
 // testet die Klasse Gerade

 public static void main(String[] args) throws IOException {
 BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

 // Werte der Geraden erfragen
 String s; // String für eingegebene Zeile
 System.out.println("Steigung der Geraden:");
 s = in.readLine();
 double m = Double.parseDouble(s);
 System.out.println("y-Achsen-Abschnitt:");
 s = in.readLine();
 double b = Double.parseDouble(s);

 // Verschiebungsvektor erfragen
 System.out.println("Verschiebungsvektor, x-Komponente:");
 s = in.readLine();
 double x = Double.parseDouble(s);
 System.out.println("Verschiebungsvektor, y-Komponente:");
 s = in.readLine();
 double y = Double.parseDouble(s);

 // Drehwinkel erfragen
 System.out.println("Drehwinkel in Grad: ");
 s = in.readLine();
 double alpha = Math.PI*Double.parseDouble(s)/180.0; // in Bogenmaß

 Gerade g1 = new Gerade(m, b);
 Gerade g2 = new Gerade(m, b); // Kopie für den Schnittpunkt

 System.out.println("Eingegebene Gerade: " + g1);
 g1.verschiebe(x, y);
 System.out.println("Verschobene Gerade: " + g1);
 g1.drehe(alpha);
 System.out.println("Verschobene und gedrehte Gerade: " + g1);
 Punkt schnittpunkt = g1.berechneSchnittpunkt(g2);
 System.out.println("Schnittpunkt erster mit letzter Geraden: "
 + schnittpunkt);
 }
}
```

## Aufgabe 8

- Implementieren Sie eine Klasse Datum mit den Feldern tag, monat, jahr und den Methoden
  - String toString() // Datum als String
  - Datum morgen(), Datum gestern() // Datum +/- 1 Tag
- Schreiben Sie dann eine Methode
  - int tageAbstand(Datum d)
- die die Zahl der Tage vom eigenen bis zum angegeben Datum zurückgibt.
- Damit sollte es nicht schwer sein, die Methode
  - public String wochentag()
- hinzuzufügen, die den Wochentag des Datums als String zurückgibt.
- Testen Sie immer alles mit einer geeigneten main-Methode.
- Hinweise:
  - Eine Routine istSchaltjahr() könnte sehr nützlich sein!
  - Die Längen der Monate schreibt man am einfachsten in ein Array, ebenso die Namen der Monate und Wochentage.
  - Schreiben Sie eine Hilfs-Routine, die die Anzahl der Tage bis zu einem festen Datum (etwa seit Christi Geburt) bestimmt.
- lauffähiges Applet
- Source



# Datum.java

```
import java.io.*;

public class Datum {

 // Datenfelder

 private int tag; // Werte von 1 bis 31
 private int monat; // Werte von 1 bis 12
 private int jahr; // Werte natürlich in voller Stellenzahl

 // Hilfsfelder

 // kleiner Trick: Dummy-Wert für 0, um direkt mit monat arbeiten zu können
 private String[] monatsName = {"", "Januar", "Februar", "März", "April",
 "Mai", "Juni", "Juli", "August", "September",
 "Oktober", "November", "Dezember"};

 private int[] monatsLänge = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

 // Tage vom 1.1. bis zum 1. eines Monats
 private int[] monatstageSeitNeujahr = {-1, 0, 31, 59, 90, 120, 151,
 181, 212, 243, 273, 304, 334};

 private String[] tagesName = {"Montag", "Dienstag", "Mittwoch",
 "Donnerstag", "Freitag", "Samstag", "Sonntag"};

 public Datum(int t, int m, int j) {
 // gleich prüfen, bei Fehler Standardwerte (1.1.)
 // besser: Ausnahme auslösen

 jahr = j;

 if ((m >= 1) && (m <= 12)) {
 monat = m;
 } else {
 monat = 1;
 }

 if ((t >= 1) && (t <= monatsLänge(m, j))) {
 tag = t;
 } else {
 tag = 1;
 }
 }

 public String toString() {
 // Datum als String
 return tag + ". " + monatsName[monat] + " " + jahr;
 }

 public String wochentag() {
 // gibt den Wochentag des aktuellen Datums zurück
 return tagesName[tageSeitCG(this) % 7];
 }

 public Datum morgen() {
 if (tag + 1 <= monatsLänge(monat, jahr)) { // gleicher Monat
 return new Datum(tag + 1, monat, jahr);
 }
 }
}
```

```

 } else if (monat + 1 <= 12) { // Monatswechsel, gleiches Jahr
 return new Datum(1, monat + 1, jahr);
 } else { // Jahreswechsel
 return new Datum(1, 1, jahr + 1);
 }
}

public Datum gestern() {
 if (tag >= 2) { // gleicher Monat
 return new Datum(tag - 1, monat, jahr);
 } else if (monat >= 2) { // Monatswechsel, gleiches Jahr
 return new Datum(monatsLänge(monat - 1, jahr), monat - 1, jahr);
 } else { // Jahreswechsel
 return new Datum(31, 12, jahr - 1);
 }
}

public int tageAbstand(Datum d) {
 return tageSeitCG(d) - tageSeitCG(this);
}

private boolean istSchaltjahr(int j) {
 return (j % 4 == 0) && ((j % 100 != 0) || (j % 400 == 0));
}

private int monatsLänge(int m, int j) {
 // gibt Zahl der Tage des Monats m im Jahr j zurück

 if ((m == 2) && istSchaltjahr(j)) {
 return 29;
 } else {
 return monatsLänge[m];
 }
}

private int tageSeitCG(Datum d) {
 // Tage vom 1.1.1 bis zum Datum d
 // funktioniert so nur für positive Jahreszahlen
 // (eigentlich sogar erst seit der Gregorianischen Kalenderreform)

 int tageSeitNeujahr = monatstageSeitNeujahr[d.monat] + d.tag - 1;
 if (istSchaltjahr(d.jahr) && (d.monat > 2)) {
 tageSeitNeujahr++;
 }

 // berechne Schalttage vom Jahr 1 bis zum angegebenen Jahr
 // (ausschliesslich dem Jahr selber)
 int schalttage = 0;
 for (int j = 1; j < d.jahr; j++) {
 if (istSchaltjahr(j)) {
 schalttage++;
 }
 }
 return 365*(d.jahr - 1) + schalttage + tageSeitNeujahr;
}

static public void main(String[] args) throws IOException {
 BufferedReader in = new BufferedReader(

```

```

 new InputStreamReader(System.in));

// hole heutiges Datum vom Benutzer
String s; // String für eingegebene Zeile
System.out.println("Datum von heute eingeben");
System.out.println("Tag:");
s = in.readLine();
int t = Integer.parseInt(s);
System.out.println("Monat:");
s = in.readLine();
int m = Integer.parseInt(s);
System.out.println("Jahr:");
s = in.readLine();
int j = Integer.parseInt(s);

Datum heute = new Datum(t, m, j);
System.out.println("Heute ist " +
 heute.wochentag() + ", der " + heute);

System.out.println("Morgen ist der " + heute.morgen());
System.out.println("Gestern war der " + heute.gestern());

// hole weiteres Datum vom Benutzer
System.out.println("weiteres Datum eingeben");
System.out.println("Tag:");
s = in.readLine();
int t2 = Integer.parseInt(s);
System.out.println("Monat:");
s = in.readLine();
int m2 = Integer.parseInt(s);
System.out.println("Jahr:");
s = in.readLine();
int j2 = Integer.parseInt(s);

Datum demnächst = new Datum(t2, m2, j2);

System.out.println("Von " + heute + " bis " + demnächst + " sind es " +
 heute.tageAbstand(demnächst) + " Tage.");
}
}

```

## Aufgabe 8a

- Entwickeln Sie eine Klasse QuadratischeGleichung und implementieren Sie geeignete Methoden zur Erzeugung und Ausgabe sowie zum Berechnen der Lösungen. Denken Sie dabei auch an Spezialfälle.
- Hinweise:
  - Datenfelder sind hier am einfachsten die drei Koeffizienten a, b, c der quadratischen Gleichung.
  - Eine gute Ausgabefunktion sollte auch die Vorzeichen richtig darstellen, also nicht einfach
    - $x + -5.0$
  - Besondere Beachtung benötigt die Lösung des Falles  $a = 0$ .
- lauffähiges Applet
- Source

# QuadratischeGleichung.java

```
import java.io.*;

public class QuadratischeGleichung {
 // beschreibt $a*x^2 + b*x + c = 0$

 double a;
 double b;
 double c;

 public QuadratischeGleichung(double a1, double b1, double c1) {
 // erzeuge neue Gleichung mit gegebenen Koeffizienten
 a = a1;
 b = b1;
 c = c1;
 }

 private String toSignedString(double d) {
 // erzeugt String mit explizitem Vorzeichen
 if (d >= 0) {
 return("+ " + d);
 } else {
 return("- " + (-d));
 }
 }

 public String toString() {
 // erzeugt Textdarstellung der quadratischen Gleichung
 String s = a + "*x^2 " + toSignedString(b) + "*x "
 + toSignedString(c);
 return s;
 }

 private double discriminant() {
 // die Diskriminante der Gleichung
 return(b*b/(4*a*a) - c/a);
 }

 private double[] getLinearSolution() {
 // gib die Lösung der linearen Gleichung zurück
 double[] result;

 if (b != 0.0) {
 // Normalfall der linearen Gleichung
 result = new double[1];
 result[0] = -c/b;
 } else if (c != 0) {
 // Gleichung lautet einfach $c = 0$, aber $c \neq 0$
 // Widerspruch, also keine Lösungen
 result = new double[0];
 } else {
 // b und c sind 0, die Gleichung lautet $0*x = 0$
 // alle Zahlen sind Lösung
 // man kann aber nicht alle Zahlen zurückgeben!
 // Notlösung: eine Lösung ausgeben und Warnungsmeldung dazu

 System.out.println("WARNUNG: Gleichung $0*x = 0$, alle x sind Lösung!");
 result = new double[1];
 result[0] = 1.0;
 }
 }
}
```

```

 }
 return result;
}

public double[] getSolution() {
 // gibt Vektor mit den Lösungen zurueck
 // berücksichtigt auch a = 0

 if (a == 0.0) {
 // lineare Gleichung, verwende entsprechende Lösungsroutine
 return getLinearSolution();
 }

 // hier ist klar: a != 0
 double a1 = -b/(2*a);
 double d = discriminant();
 double[] result;

 if (d > 0) {
 result = new double[2];
 double dRoot = Math.sqrt(d);
 result[0] = a1 + dRoot;
 result[1] = a1 - dRoot;
 } else if (d == 0) {
 result = new double[1];
 result[0] = a1;
 } else {
 // d < 0: es gibt keine (reelle) Lösung
 result = new double[0];
 }

 return(result);
}

public static void main(String[] args) throws IOException {
 BufferedReader in = new BufferedReader(
 new InputStreamReader(System.in));

 QuadratischeGleichung qe1 = null;

 // hole die Koeffizienten vom Benutzer
 String s; // String für eingegebene Zeile
 System.out.println("Koeffizient von x^2 eingeben:");
 s = in.readLine();
 double a1 = Double.parseDouble(s);
 System.out.println("Koeffizient von x eingeben:");
 s = in.readLine();
 double b1 = Double.parseDouble(s);
 System.out.println("Konstante eingeben:");
 s = in.readLine();
 double c1 = Double.parseDouble(s);

 qe1 = new QuadratischeGleichung(a1, b1, c1);

 System.out.println("Die quadratische Gleichung " + qe1);

 double[] result = qe1.getSolution();
 int count = result.length;
 System.out.println("hat " + count + " Lösungen");

 if (count > 0) {
 System.out.println(result[0]);
 }
}

```

```
}
if (count > 1) {
 System.out.println(result[1]);
}
}
}
```

## Aufgabe 9

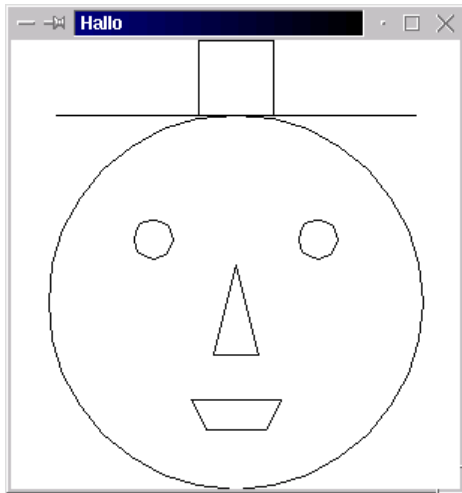
- Ziel dieser Aufgabe ist ein erster Rohbau für ein primitives Zeichenprogramm, das einfache Figuren manipulieren und ausgeben kann. Das Programm soll mit folgenden Figuren umgehen können:
  - Linie, Dreieck, Quadrat, Polygon, Kreis
- Die grundlegende Klasse Punkt beschreibe sowohl einen Punkt auf der Zeichenfläche als auch einen Verschiebungsvektor. Folgende Methoden sollen für die geeigneten Figuren implementiert werden:

move(Punkt p)	verschiebe um Vektor p
scale(double d)	vergrößere/verkleinere um Faktor d
rotate(double u)	drehe um Winkel u um den Ursprung
String toString()	Objektyp und Koordinaten ausgeben

- Entwerfen Sie zunächst einen Stammbaum: Welche Klassen erben von welchen? Welche abstrakten Basisklassen oder Interfaces kann man sinnvoll einführen? Wo werden welche Methoden definiert bzw. implementiert?
- Legen Sie als nächstes fest, welche internen Daten die einzelnen Figuren benötigen. Beachten Sie dabei den Stammbaum bzw. ändern Sie ihn ggf. ab.
- Überlegen Sie nun, welche Konstruktoren und Hilfsmethoden Sie brauchen, um alle Figuren erzeugen oder auch abändern zu können.
- Implementieren Sie dann alle Methoden für alle Figuren. Tip:
  - Rotieren eines 2d-Punkts um Winkel  $\alpha$  um Ursprung mit Dreh-Matrizen:
    - $$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos \alpha & \sin \alpha \\ -\sin \alpha & \cos \alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$
    - oder ausgeschrieben:
 
$$x' = x \cos u + y \sin u$$

$$y' = -x \sin u + y \cos u$$
  - Verwenden Sie Math.cos(), Math.sin().
- Testen Sie Ihre Klassen, indem Sie ein Programm schreiben, das die untenstehende zusammengesetzte Figur erzeugt und mit Hilfe der toString-Methoden der Einzelteile das Ergebnis ausgibt.

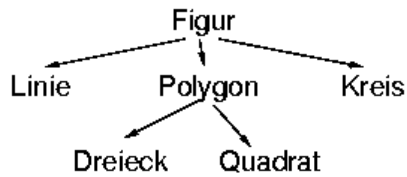




- lauffähiges Applet
- weitere Hinweise

## Weitere Hinweise zu Aufgabe 9

- Gemeinsame Oberklasse ist Figur, Stammbaum:



- Alle vier Methoden sind für alle Figuren sinnvoll, können aber nicht in Figur implementiert werden. Daher ist Figur abstrakt. Da es überhaupt keine von Figur zu ererbenden Dinge gibt, kann es als Schnittstelle eingeführt werden.
- Alle vier Operationen sind auch bei Punkt sinnvoll, vor allem zur Implementierung der Operationen in den Figuren. Ob man Punkt deswegen in den Kreis der Figuren aufnimmt, ist an dieser Stelle sicher Geschmackssache. In Aufgabe 11 kommt aber die Methode draw für alle Figuren hinzu. Hier zeigt sich, dass Punkt nur ein Bestandteil einer Figur ist: er hat keine draw-Methode. Daher wird er hier als unabhängig von Figur gesetzt.
- Alle Funktionen von Dreieck und Quadrat können auch direkt in Polygon implementiert werden, indem man sie als Spezialfälle von Polygon betrachtet.
- Um bei toString einen Namen ("Quadrat", "Polygon") benutzen zu können, wird eine Methode getName() in Polygon eingeführt und in toString verwendet. Quadrat und Dreieck überschreiben einfach die geerbte Funktion.
- Alles weitere kann direkt den Quellen entnommen werden:
  - Punkt.java, Figur.java, Linie.java, Kreis.java,
  - Polygon.java, Dreieck.java, Quadrat.java
  - TestFigur.java

# Punkt.java

```
public class Punkt {
 // stellt einen Punkt auf einer abstrakten 2d-Zeichenfläche dar
 // wird gleichzeitig auch als Verschiebungsvektor verwendet, ist also
 // als Ortsvektor zu verstehen

 // abstrakte Zeichenfläche:
 // unendlich ausgedehnt
 // genaue Koordinaten (keine Pixel)

 protected double x, y;

 // Konstruktoren

 public Punkt(double x1, double y1) {
 x = x1;
 y = y1;
 }

 public Punkt() {
 // Standardpunkt: der Ursprung
 this(0.0, 0.0);
 }

 // öffentliche Methoden

 public void move(Punkt p) {
 // um p verschieben
 x += p.x;
 y += p.y;
 }

 public void scale(double s) {
 // skaliert um Faktor s, Bezugspunkt 0
 // meint hier: als Vektor vom Ursprung
 x *= s;
 y *= s;
 }

 public void rotate(double alpha) {
 // dreht um Winkel alpha, Drehpunkt 0

 double xn = x * Math.cos(alpha) + y*Math.sin(alpha);
 y = -x * Math.sin(alpha) + y*Math.cos(alpha);
 x = xn;
 }

 public String toString() {
 return "(" + x + ", " + y + ")";
 }
}
```

## Figur.java

```
public interface Figur {
 // Oberklasse der Zeichenobjekte
 // kann nichts implementieren, daher am besten Interface

 public void move(Punkt p); // verschiebt um "Vektor" p
 public void scale(double s); // skaliert um Faktor s, Bezugspunkt O
 public void rotate(double alpha); // dreht um Winkel alpha, Drehpunkt O
 public String toString(); // beschreibt Objekt als String
}
```

# Linie.java

```
public class Linie implements Figur {
 // einfache Figur, gegeben durch zwei Punkte

 private Punkt p1, p2;

 // Konstruktoren

 public Linie(Punkt pa, Punkt pb) {
 p1 = pa;
 p2 = pb;
 }

 public Linie() {
 // Standardlinie: auf der x-Ache von 0 bis 1
 this(new Punkt(), new Punkt(1.0, 0.0));
 }

 // öffentliche Methoden

 public void move(Punkt p) {
 // verschiebt um "Vektor" p

 p1.move(p);
 p2.move(p);
 }

 public void scale(double s) {
 // skaliert um Faktor s, Bezugspunkt 0

 p1.scale(s);
 p2.scale(s);
 }

 public void rotate(double alpha) {
 // dreht um Winkel alpha, Drehpunkt 0

 p1.rotate(alpha);
 p2.rotate(alpha);
 }

 public String toString() {
 // beschreibt Objekt als String

 return "LINIE\n" + " " + p1 + "\n " + p2 + "\n";
 }
}
```

# Kreis.java

```
public class Kreis implements Figur {
 // Kreis, gegeben durch Mittelpunkt und Radius

 private Punkt m; // Mittelpunkt
 private double r; // Radius

 // Konstruktoren

 public Kreis(Punkt p, double d) {
 m = p;
 r = d;
 }

 public Kreis() {
 // Standardkreis: um Ursprung, Radius 1
 this(new Punkt(), 1.0);
 }

 // öffentliche Methoden

 public void move(Punkt p) {
 // verschiebt um "Vektor" p

 m.move(p);
 }

 public void scale(double s) {
 // skaliert um Faktor s, Bezugspunkt O

 m.scale(s);
 r *= s;
 }

 public void rotate(double alpha) {
 // dreht um Winkel alpha, Drehpunkt O

 m.rotate(alpha);
 }

 public String toString() {
 // beschreibt Objekt als String

 return "KREIS\n" + " " + "r = " + r + "\n" + " " + m + "\n";
 }
}
```

# Polygon.java

```
public class Polygon implements Figur {
 // gegeben durch Array von Punkten

 protected Punkt[] p;

 // Konstruktoren

 public Polygon(Punkt[] p1) {
 p = p1;
 }

 public Polygon() {
 // "leeres" Polygon, wird für Vererbung gebraucht
 p = null;
 }

 // öffentliche Methoden

 public void move(Punkt p1) {
 // verschiebt um "Vektor" p1

 for (int i = 0; i < p.length; i++) {
 p[i].move(p1);
 }
 }

 public void scale(double s) {
 // skaliert um Faktor s, Bezugspunkt O

 for (int i = 0; i < p.length; i++) {
 p[i].scale(s);
 }
 }

 public void rotate(double alpha) {
 // dreht um Winkel alpha, Drehpunkt O

 for (int i = 0; i < p.length; i++) {
 p[i].rotate(alpha);
 }
 }

 public String toString() {
 // beschreibt Objekt als String

 String str = getName() + "\n";

 for (int i = 0; i < p.length; i++) {
 str += " " + p[i] + "\n";
 }

 return str;
 }

 protected String getName() {
 // wird von abgeleiteten Klassen überschrieben
 return "POLYGON";
 }
}
```

# Dreieck.java

```
public class Dreieck extends Polygon {
 // gegeben durch Array von 3 Punkten

 // Konstruktor

 public Dreieck(Punkt p1, Punkt p2, Punkt p3) {
 p = new Punkt[] {p1, p2, p3};
 }

 public Dreieck() {
 // Standarddreieck
 this(new Punkt(), new Punkt(1.0, 0.0), new Punkt(0.0, 1.0));
 }

 protected String getName() {
 // überschreibt Methode in Polygon
 return "DREIECK";
 }

 // öffentliche Methoden kommen von Polygon
}
```



# Quadrat.java

```
public class Quadrat extends Polygon {
 // gegeben durch Array von 4 Punkten

 // Konstruktoren

 public Quadrat() {
 // Standardquadrat von (0,0) bis (1,1)
 super(new Punkt[] {new Punkt(), new Punkt(1.0, 0.0),
 new Punkt(1.0, 1.0), new Punkt(0.0, 1.0)});
 }

 public Quadrat(double l, double alpha, Punkt p) {
 // Quadrat der Seitenlänge l, gekippt um alpha, untere Ecke bei p
 this(); // Standardquadrat
 scale(l);
 rotate(alpha);
 move(p);
 }

 protected String getName() {
 // überschreibt Methode in Polygon
 return "QUADRAT";
 }

 // öffentliche Methoden kommen von Polygon
}
```

# TestFigur.java

```
public class TestFigur {

 protected Figur[] figur; // Liste aller zu malenden Figuren

 // Konstruktor
 public TestFigur() {
 // erzeuge Liste aus Standard-Figuren
 figur = new Figur[7];

 // Gesicht
 figur[0] = new Kreis(new Punkt(150.0, 175.0), 125.0);

 // Nase
 figur[1] = new Dreieck(new Punkt(5.0, 0.0),
 new Punkt(0.0, 20.0), new Punkt(10.0, 20.0));
 figur[1].scale(3.0);
 figur[1].move(new Punkt(135.0, 150.0));

 // Hut
 figur[2] = new Quadrat(50.0, 0.0, new Punkt(125.0, 0.0));

 // Augen
 figur[3] = new Kreis(new Punkt(95.0, 132.5), 12.5);
 figur[4] = new Kreis(new Punkt(205.0, 132.5), 12.5);

 // Hutkrempe
 figur[5] = new Linie(new Punkt(30.0, 50.0), new Punkt(270.0, 50.0));

 // Mund
 Punkt[] p = {new Punkt(), new Punkt(5.0, 10.0),
 new Punkt(25.0, 10.0), new Punkt(30.0, 0.0)};
 figur[6] = new Polygon(p);
 figur[6].scale(2.0);
 figur[6].move(new Punkt(120.0, 240.0));
 }

 public static void main(String[] args) {

 // erzeuge eine TestFigur
 TestFigur tf = new TestFigur();

 // drucke die Beschreibung aller Figuren
 for (int i = 0; i < tf.figur.length; i++) {
 System.out.println(tf.figur[i].toString());
 }
 }
}
```

## Aufgabe 10

- Schreiben Sie ein Programm, das eine Datei einliest und die Zahl der Zeichen, Wörter und Zeilen in dieser Datei ausgibt.
- Source

# Lösung von Aufgabe 10

```
import java.io.*;

public class CountFile {

 public static void main(String[] args) {

 String eingabeDatei = args[0];

 int nZeichen = 0;
 int nWoerter = 0;
 int nZeilen = 0;

 try {
 BufferedReader in = new BufferedReader(new FileReader(eingabeDatei));
 int c; // enthält gelesenes Zeichen oder -1

 while ((c = in.read()) != -1) {
 nZeichen++;
 if (c == ' ') {
 nWoerter++;
 } else if (c == '\n') {
 nZeilen++;
 nWoerter++;
 }
 }
 in.close();
 }
 catch(IOException e) {
 System.out.println("Fehler beim Kopieren:");
 e.printStackTrace(System.out);
 }

 nWoerter++; // ein Wert mehr als Zwischenräume
 // nicht für nZeilen: nur ganze Zeilen mit Newline zählen mit

 System.out.println("Zahl der Zeichen: " + nZeichen);
 System.out.println("Zahl der Wörter: " + nWoerter);
 System.out.println("Zahl der Zeilen: " + nZeilen);
 }
}
```

# Aufgabe 11

- Ergänzen Sie die Klassen von Aufgabe 9 jeweils um eine Methode
  - void draw(Graphics g)
- die die jeweilige Figur malt.
- Schreiben Sie ein Applet, das diese Klassen verwendet, um die Beispielfigur auch graphisch auszugeben und erstellen Sie eine entsprechende HTML-Seite.
- Hinweise:
  - Zum Zeichnen muss die Beziehung zwischen den Punkt-Koordinaten der Figuren und den Pixel-Koordinaten geklärt werden. Am einfachsten setzt man
    - Pixel = int(Punkt)
  - und kümmert sich nicht weiter um Ausschnitt und Skalierungen.
  - Entsprechende Routinen zum Umrechnen (hier natürlich sehr einfach) lassen sich gut als Methoden getX(), getY() von Punkt unterbringen.
- lauffähiges Applet
- Quellen aller Klassen:
  - farbig markiert sind Änderungen zu den Quellen von Aufgabe 9, bei DemoFigur.java im Vergleich zu TestFigur.java
  - Punkt.java, Figur.java, Linie.java, Kreis.java, Polygon.java, Dreieck.java, Quadrat.java, DemoFigur.java, DemoFigurApplet.java

# Punkt.java

```
public class Punkt {
 // stellt einen Punkt auf einer abstrakten 2d-Zeichenfläche dar
 // wird gleichzeitig auch als Verschiebungsvektor verwendet, ist also
 // als Ortsvektor zu verstehen

 // abstrakte Zeichenfläche:
 // unendlich ausgedehnt
 // genaue Koordinaten (keine Pixel)

 protected double x, y;

 // Konstruktoren

 public Punkt(double x1, double y1) {
 x = x1;
 y = y1;
 }

 public Punkt() {
 // Standardpunkt: der Ursprung
 this(0.0, 0.0);
 }

 // öffentliche Methoden

 public void move(Punkt p) {
 // um p verschieben
 x += p.x;
 y += p.y;
 }

 public void scale(double s) {
 // skaliert um Faktor s, Bezugspunkt 0
 // meint hier: als Vektor vom Ursprung
 x *= s;
 y *= s;
 }

 public void rotate(double alpha) {
 // dreht um Winkel alpha, Drehpunkt 0

 double xn = x * Math.cos(alpha) + y*Math.sin(alpha);
 y = -x * Math.sin(alpha) + y*Math.cos(alpha);
 x = xn;
 }

 public int getX() {
 // gibt die X-Koordinate des Punkts für den Bildschirm
 return (int) Math.round(x);
 }

 public int getY() {
 // gibt die Y-Koordinate des Punkts für den Bildschirm
 return (int) Math.round(y);
 }
}
```

```
public String toString() {
 return "(" + x + ", " + y + ")";
}
}
```

# Figur.java

```
import java.awt.*;

public interface Figur {
 // Oberklasse der Zeichenobjekte
 // kann nichts implementieren, daher am besten Interface

 public void move(Punkt p); // verschiebt um "Vektor" p
 public void scale(double s); // skaliert um Faktor s, Bezugspunkt O
 public void rotate(double alpha); // dreht um Winkel alpha, Drehpunkt O
 public void draw(Graphics g); // zeichne die Figur
 public String toString(); // beschreibt Objekt als String
}
```



# Linie.java

```
import java.awt.*;

public class Linie implements Figur {
 // einfache Figur, gegeben durch zwei Punkte

 private Punkt p1, p2;

 // Konstruktoren

 public Linie(Punkt pa, Punkt pb) {
 p1 = pa;
 p2 = pb;
 }

 public Linie() {
 // Standardlinie: auf der x-Ache von 0 bis 1
 this(new Punkt(), new Punkt(1.0, 0.0));
 }

 // öffentliche Methoden

 public void move(Punkt p) {
 // verschiebt um "Vektor" p

 p1.move(p);
 p2.move(p);
 }

 public void scale(double s) {
 // skaliert um Faktor s, Bezugspunkt 0

 p1.scale(s);
 p2.scale(s);
 }

 public void rotate(double alpha) {
 // dreht um Winkel alpha, Drehpunkt 0

 p1.rotate(alpha);
 p2.rotate(alpha);
 }

 public void draw(Graphics g) {
 // zeichne die Figur

 g.drawLine(p1.getX(), p1.getY(), p2.getX(), p2.getY());
 }

 public String toString() {
 // beschreibt Objekt als String

 return "LINIE\n" + " " + p1 + "\n " + p2 + "\n";
 }
}
```

# Kreis.java

```
import java.awt.*;

public class Kreis implements Figur {
 // Kreis, gegeben durch Mittelpunkt und Radius

 private Punkt m; // Mittelpunkt
 private double r; // Radius

 // Konstruktoren

 public Kreis(Punkt p, double d) {
 m = p;
 r = d;
 }

 public Kreis() {
 // Standardkreis: um Ursprung, Radius 1
 this(new Punkt(), 1.0);
 }

 // öffentliche Methoden

 public void move(Punkt p) {
 // verschiebt um "Vektor" p

 m.move(p);
 }

 public void scale(double s) {
 // skaliert um Faktor s, Bezugspunkt O

 m.scale(s);
 r *= s;
 }

 public void rotate(double alpha) {
 // dreht um Winkel alpha, Drehpunkt O

 m.rotate(alpha);
 }

 public void draw(Graphics g) {
 // zeichne die Figur

 int iRadius = (int) Math.round(r);
 g.drawOval(m.getX() - iRadius, m.getY() - iRadius, 2*iRadius, 2*iRadius);
 }

 public String toString() {
 // beschreibt Objekt als String

 return "KREIS\n" + " " + "r = " + r + "\n" + " " + m + "\n";
 }
}
```

# Polygon.java

```
import java.awt.*;

public class Polygon implements Figur {
 // gegeben durch Array von Punkten

 protected Punkt[] p;

 // Konstruktoren

 public Polygon(Punkt[] p1) {
 p = p1;
 }

 public Polygon() {
 // "leeres" Polygon, wird für Vererbung gebraucht
 p = null;
 }

 // öffentliche Methoden

 public void move(Punkt p1) {
 // verschiebt um "Vektor" p1

 for (int i = 0; i < p.length; i++) {
 p[i].move(p1);
 }
 }

 public void scale(double s) {
 // skaliert um Faktor s, Bezugspunkt 0

 for (int i = 0; i < p.length; i++) {
 p[i].scale(s);
 }
 }

 public void rotate(double alpha) {
 // dreht um Winkel alpha, Drehpunkt 0

 for (int i = 0; i < p.length; i++) {
 p[i].rotate(alpha);
 }
 }

 public void draw(Graphics g) {
 // zeichne die Figur

 // erzeuge int-Arrays für x- und y-Koordinaten für drawPolygon
 int[] xPoints = new int[p.length];
 int[] yPoints = new int[p.length];

 for (int i = 0; i < p.length; i++) {
 xPoints[i] = p[i].getX();
 yPoints[i] = p[i].getY();
 }

 g.drawPolygon(xPoints, yPoints, p.length);
 }

 public String toString() {
```

```
// beschreibt Objekt als String

String str = getName() + "\n";

for (int i = 0; i < p.length; i++) {
 str += " " + p[i] + "\n";
}

return str;
}

protected String getName() {
 // wird von abgeleiteten Klassen überschrieben
 return "POLYGON";
}
}
```

# Dreieck.java

```
public class Dreieck extends Polygon {
 // gegeben durch Array von 3 Punkten

 // Konstruktor

 public Dreieck(Punkt p1, Punkt p2, Punkt p3) {
 p = new Punkt[] {p1, p2, p3};
 }

 public Dreieck() {
 // Standarddreieck
 this(new Punkt(), new Punkt(1.0, 0.0), new Punkt(0.0, 1.0));
 }

 protected String getName() {
 // überschreibt Methode in Polygon
 return "DREIECK";
 }

 // öffentliche Methoden kommen von Polygon
}
```

# Quadrat.java

```
public class Quadrat extends Polygon {
 // gegeben durch Array von 4 Punkten

 // Konstruktoren

 public Quadrat() {
 // Standardquadrat von (0,0) bis (1,1)
 super(new Punkt[] {new Punkt(), new Punkt(1.0, 0.0),
 new Punkt(1.0, 1.0), new Punkt(0.0, 1.0)});
 }

 public Quadrat(double l, double alpha, Punkt p) {
 // Quadrat der Seitenlänge l, gekippt um alpha, untere Ecke bei p
 this(); // Standardquadrat
 scale(l);
 rotate(alpha);
 move(p);
 }

 protected String getName() {
 // überschreibt Methode in Polygon
 return "QUADRAT";
 }

 // öffentliche Methoden kommen von Polygon
}
```

# DemoFigur.java

```
import java.awt.*;
import javax.swing.*;

public class DemoFigur extends JPanel {
 // Zeichenfläche (JPanel) mit einer einfachen Demo-Figur

 private Figur[] figur; // Liste aller zu malenden Figuren

 // Konstruktor
 public DemoFigur() {
 // erzeuge Liste aus Standard-Figuren
 figur = new Figur[7];

 // Gesicht
 figur[0] = new Kreis(new Punkt(150.0, 175.0), 125.0);

 // Nase
 figur[1] = new Dreieck(new Punkt(5.0, 0.0),
 new Punkt(0.0, 20.0), new Punkt(10.0, 20.0));
 figur[1].scale(3.0);
 figur[1].move(new Punkt(135.0, 150.0));

 // Hut
 figur[2] = new Quadrat(50.0, 0.0, new Punkt(125.0, 0.0));

 // Augen
 figur[3] = new Kreis(new Punkt(95.0, 132.5), 12.5);
 figur[4] = new Kreis(new Punkt(205.0, 132.5), 12.5);

 // Hutkrempe
 figur[5] = new Linie(new Punkt(30.0, 50.0), new Punkt(270.0, 50.0));

 // Mund
 Punkt[] p = {new Punkt(), new Punkt(5.0, 10.0),
 new Punkt(25.0, 10.0), new Punkt(30.0, 0.0)};
 figur[6] = new Polygon(p);
 figur[6].scale(2.0);
 figur[6].move(new Punkt(120.0, 240.0));

 // weißen Hintergrund
 setBackground(Color.white); // geerbt von JPanel
 }

 public void paintComponent(Graphics g) {
 // male den Hintergrund (erledigt die Mutter JPanel)
 super.paintComponent(g);

 // zeichne alle Figuren
 for (int i = 0; i < figur.length; i++) {
 figur[i].draw(g);
 }
 }
}
```

# DemoFigurApplet.java

```
import java.awt.*;
import javax.swing.*;

public class DemoFigurApplet extends JApplet {

 public void init() {
 // besorge ContentPane
 Container p = getContentPane();

 // packe eine DemoFigur hinein
 DemoFigur hw = new DemoFigur();
 p.add(hw);
 }
}
```



## Aufgabe 12

- Schreiben Sie ein Applet, mit dem bunte Drei-, Vier- und Fünfecke mit zufälligen Eckpunkten und Farben auf einer Zeichenfläche dargestellt werden. Als Eingabeelemente sollen eine ComboBox zur Auswahl der Eckenanzahl, ein Button zum Hinzufügen eines Polygons und einer zum Löschen der Zeichenfläche dienen.
- Verwenden Sie als Ausgangspunkt die Figuren aus Aufgabe 11.
- lauffähiges Applet

## Aufgabe 13

- Erweitern Sie das Applet aus Aufgabe 12 zum Kaleidoskop, indem Sie
  - nur Zufallspunkte innerhalb eines 60°-Sektors zulassen und durch Spiegelungen eine sechsfache Symmetrie erzeugen,
  - die Polygone durchscheinend zeichnen, so dass später hinzugefügte grosse Polygone nicht alle alten völlig überdecken.
- Hinweise:
  - Zur Berechnung von Spiegelungen und Drehungen ist eine Methode zur Drehung um einen beliebig vorgegeben Punkt q nützlich gemäß

- $$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos \alpha & \sin \alpha \\ -\sin \alpha & \cos \alpha \end{pmatrix} \left[ \begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} q_x \\ q_y \end{pmatrix} \right] + \begin{pmatrix} q_x \\ q_y \end{pmatrix}$$

- Um durchscheinende Farben zu erzeugen, muss man beim Zeichnen eines Polygons neben der Farbe noch den Überlappungsgrad (alpha-Wert) angeben. Dazu muss der Graphics-Kontext in die neue Graphics2D-Form gebracht werden mit

- ```
// zeichne die Figur mit Graphics2D-Methoden
Graphics2D g2 = (Graphics2D) g;
```

- Das Zeichnen geschieht dann z.B. in folgender Weise

- ```
// setze Farbe und transparente Überlappung und zeichne
AlphaComposite ac =
 AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 0.6f);
g2.setComposite(ac);
g2.setColor(col);
g2.fillPolygon(xPoints, yPoints, p.length);
```

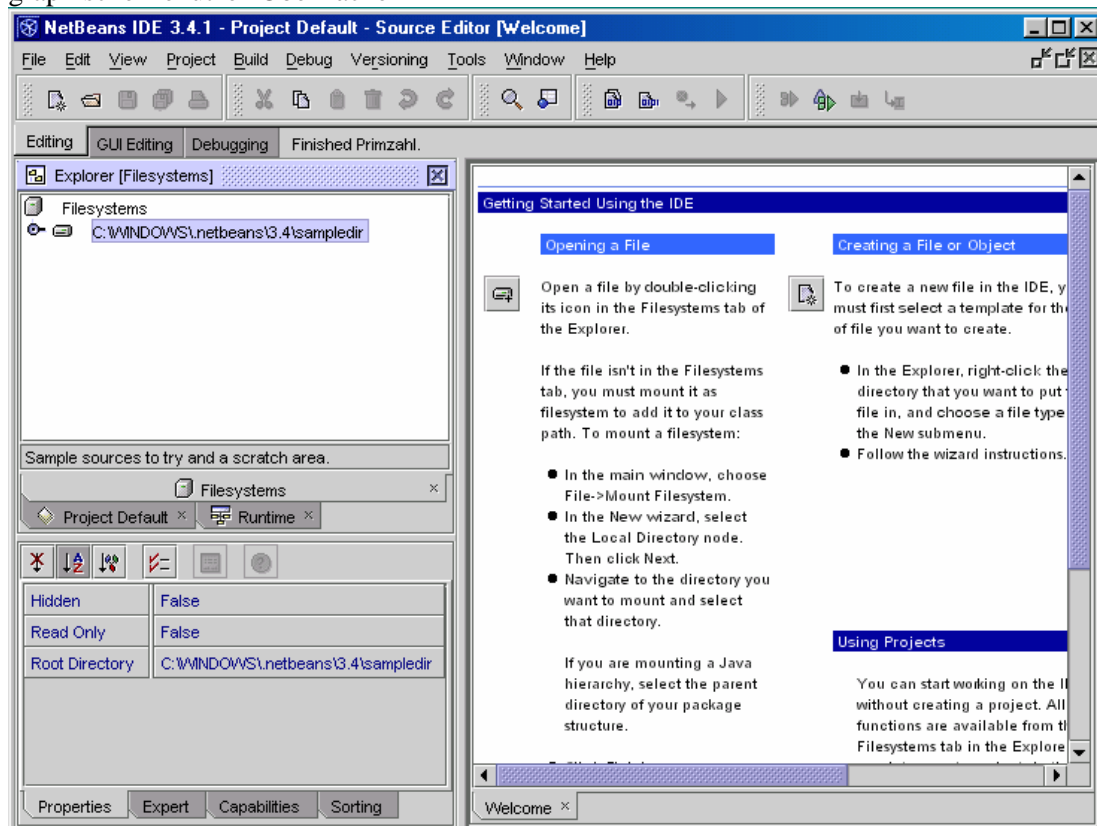
- lauffähiges Applet

# Anhang

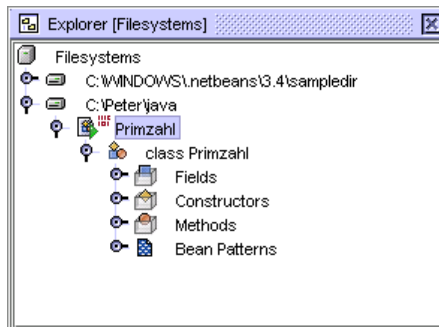
- Arbeiten mit NetBeans
- Prüfungsleistung
- Literatur
- Sourcen
- Applets



# Arbeiten mit NetBeans

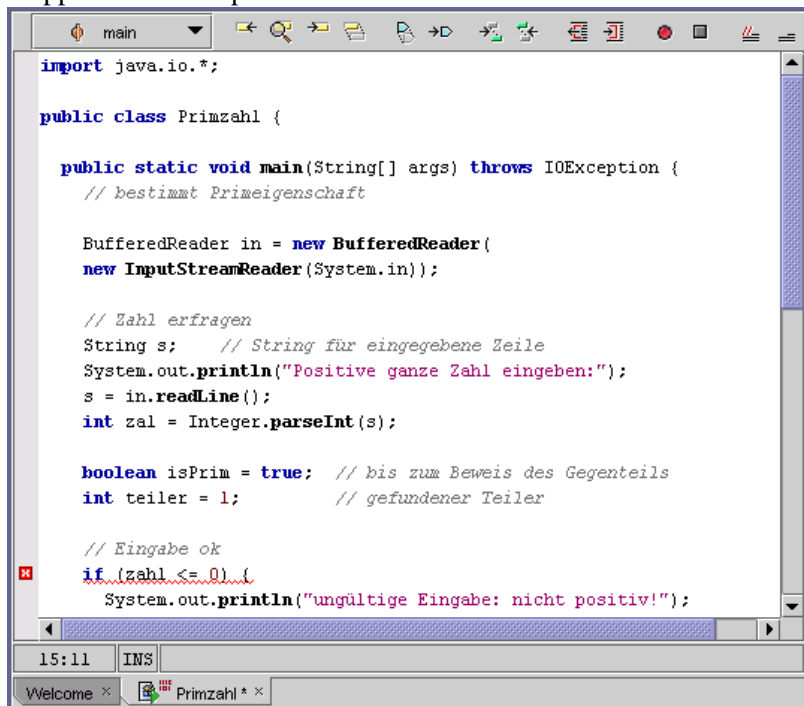
- NetBeans:
  - Entwicklungsumgebung (IDE) für Java-Programmierer (u.a.)
  - Open-Source-Projekt, frei verfügbar für viele Systeme bei [www.netbeans.org](http://www.netbeans.org)
  - komfortabler Editor
  - Integration von Compiler, Debugger und vielen weiteren Werkzeugen
  - unterstützt graphisches Oberflächendesign und Komponentenentwicklung mit JavaBeans
- Vorbereitungen:
  - Verzeichnis W:\java anlegen
  - Datei Primzahl.java hineinkopieren
- NetBeans starten:
  - graphische Benutzer-Oberfläche




- links Explorer-Bereich mit hierarchischer Darstellung aller Objekte
- rechts Editor-Bereich mit Quelltexten
- Verzeichnis mit den Quellen mounten:
  - Symbol Filesystems im Explorer-Bereich mit rechter Maustaste anklicken
  - im Kontext-Menü auswählen: Mount/LocalDirectory
  - Verzeichnis mit den eigenen Java-Quelltexten (W:\java) auswählen
  - ⇒ im Explorer erscheint Symbol für das Verzeichnis
  - durch Anklicken die Unterbausteine öffnen ⇒

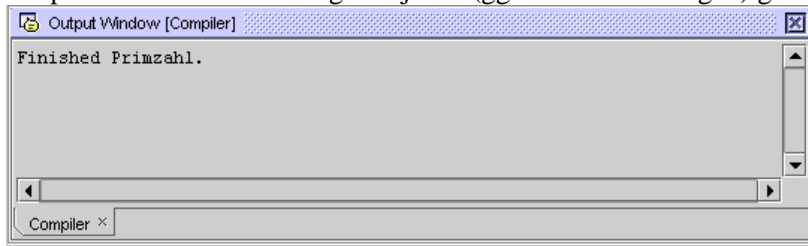


- Symbol  für Datei Primzahl.java
  - rote 0/1: kein aktuelles Class-File vorhanden
  - grüner Pfeil: enthält main-Methode
- Symbol  für Klasse Primzahl
  - darunter alle Felder, Konstruktoren und Methoden
  - Bean Patterns: i.w. besondere Methoden
- Bearbeiten von Primzahl.java:
  - Doppelklick im Explorer ⇒ Editorfenster erscheint



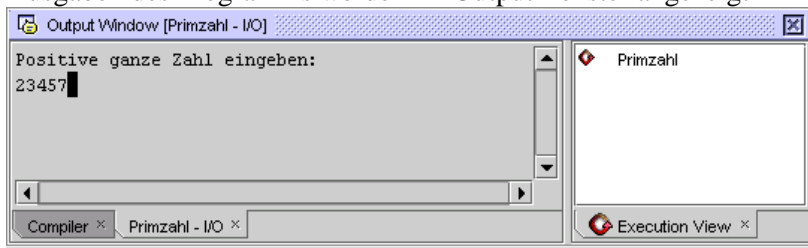
- Text mit Farben
  - Schlüsselwörter blau
  - normaler Text schwarz
  - Methodennamen fett
  - Strings magenta
  - Kommentar grau
- zeigt Fehler sofort durch Unterkringeln an
- Übersetzen und Laufenlassen:
  - Cursor im Editor-Fenster oder Primzahl.java im Explorer angeklickt
  - Build-Icon  anklicken ⇒
    - javac wird aufgerufen und Primzahl.class erzeugt

- rote Markierung an Primzahl.java verschwindet (Class-Datei ist aktuell)
- Output-Fenster mit Meldung von javac (ggf. Fehlermeldungen) geht auf



○ Execute-Icon  anklicken ⇒

- Ausgaben des Programms werden im Output-Fenster angezeigt



- dort werden auch Eingaben gemacht
- Execution-View zeigt laufendes Programm an

# Prüfungsleistung

- Erstellen Sie ein Java-Applet, das die Aufgabe 13 löst und dokumentieren Sie es.
- Abzugeben sind:
  - eine schriftliche Dokumentation,
  - ein Datenträger (CD).
- Die Dokumentation beschreibt
  - die graphische Oberfläche und Funktion des Programms aus Anwendersicht,
  - jede eigene Klasse mit
    - (öffentlicher) Schnittstelle,
    - Aufgabe der privaten Methoden und Datenfelder,
    - ggf. Erläuterungen zur Funktionsweise komplizierter Methoden,
  - Vererbungsbaum der Klassen,
  - Zusammenspiel der Objekte, ausgehend von den Benutzeraktionen auf der graphischen Oberfläche,
  - ggf. Erläuterungen zu eingesetzten und bisher nicht behandelten Java-Konstruktionen oder Standardklassen.
- Der Datenträger enthält
  - die Quellen aller verwendeten eigenen Klassen,
  - übersetzte Versionen sowie eine HTML-Seite mit funktionsfähigem Applet (zumindest unter Internet Explorer und Firefox),
  - die Dokumentation in elektronischer Form (im PDF-Format).

# Literatur

1. J. Goll, C. Weiß, F. Müller: Java als erste Programmiersprache  
Teubner, 5. Aufl. 2007, ISBN: 978-3835101470
2. J.Cowell: Essential Java 2 fast  
Springer 2000, ISBN: 1-85233-071-6
3. B. Eckel. Thinking in Java  
Prentice-Hall, 4. Aufl. 2006, ISBN: 978-0131872486  
auch online verfügbar!
4. Cay S. Horstmann, Gary Cornell: Das core Java 5-Paket. Bd.1: Grundlagen / Bd.2:  
Expertenwissen  
Addison-Wesley 2006; ISBN: 978-3827323606
5. W. Kinzel. Programmierkurs für Naturwissenschaftler und Ingenieure  
Addison-Wesley 2001, ISBN: 978-3827317797



## Sourcen

- Primzahl.java
- CopyFile.java
- HalloApplet.java
- ZeichenApplet.java
- EventApplet.java
- GridLayoutApplet.java
- BorderLayoutApplet.java
- BorderLayoutWithBorderApplet.java
- NiceLayoutApplet.java

# Primzahl.java

```
import java.io.*;

public class Primzahl {

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

 BufferedReader in = new BufferedReader(
 new InputStreamReader(System.in));

 // Zahl erfragen
 String s; // String für eingegebene Zeile
 System.out.println("Positive ganze Zahl eingeben:");
 s = in.readLine();
 int zahl = Integer.parseInt(s);

 boolean isPrim = true; // bis zum Beweis des Gegenteils
 int teiler = 1; // gefundener Teiler

 if (zahl % 2 == 0) { // durch 2 teilbar
 isPrim = false;
 teiler = 2;
 } else {
 int max = (int) Math.floor(Math.sqrt((double) zahl));
 for (int i=3; i<=max; i=i+2) {
 if (zahl % i == 0) { // durch ungerade Zahl kleiner Wurzel teilbar
 isPrim = false;
 teiler = i;
 }
 }
 }

 // Ausgabe des Ergebnisses
 if (isPrim) {
 System.out.println(zahl + " ist eine Primzahl");
 } else {
 System.out.println(zahl + " ist durch " + teiler + " teilbar");
 }
 }
}
```

# CopyFile.java

```
import java.io.*;

public class CopyFile {

 public static void main(String[] args) {

 String eingabeDatei = args[0];
 String ausgabeDatei = args[1];

 try {
 BufferedReader in =
 new BufferedReader(new FileReader(eingabeDatei));
 BufferedWriter out =
 new BufferedWriter(new FileWriter(ausgabeDatei));
 int c; // enthält gelesenes Zeichen oder -1

 while ((c = in.read()) != -1){
 out.write(c);
 }
 in.close();
 out.close();
 }
 catch(IOException e) {
 System.out.println("Fehler beim Kopieren:");
 e.printStackTrace(System.out);
 }
 }
}
```

# HalloApplet.java

```
import java.awt.*;
import javax.swing.*;

public class HalloApplet extends JApplet {

 public void init() {
 // Fenster mit ein paar Bedienungselementen ohne Funktion
 Container p = getContentPane();
 p.setLayout(new FlowLayout());

 JLabel title = new JLabel("Such was aus:");
 p.add(title);

 String[] fächer = {"Physik", "Informatik", "Thermodynamik"};
 JComboBox auswahl = new JComboBox(fächer);
 p.add(auswahl);

 JButton knopf = new JButton("Drück mich!");
 p.add(knopf);
 }
}
```

# ZeichenApplet.java

```
import java.awt.*;
import javax.swing.*;

public class ZeichenApplet extends JApplet {

 public void init() {
 Container p = getContentPane();

 // packe ein ZeichenBrett hinein
 ZeichenBrett zf = new ZeichenBrett();
 p.add(zf);
 }

 class ZeichenBrett extends JPanel {
 // Zeichenbrett als Hilfsklasse

 public void paintComponent(Graphics g) {
 // male den Hintergrund (erledigt die Mutter JPanel)
 super.paintComponent(g);

 // bestimme Größe der Zeichenfläche
 int breite = getWidth();
 int hoehe = getHeight();

 // male Text an fester Stelle hinein
 g.drawString("Hallo da draußen!", 20, 20);

 // male einen Kreis und ein Quadrat in die Mitte
 g.drawOval(breite/2 - 50, hoehe/2 - 50, 100, 100);
 g.drawRect(breite/2 - 50, hoehe/2 - 50, 100, 100);
 }
 }
}
```

# EventApplet.java

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

public class EventApplet extends JApplet {

 protected JLabel title;
 protected JComboBox selection;

 protected String[] courses = {"Physik", "Informatik", "Thermodynamik"};
 protected String[] sweets = {"Eis", "Pudding", "Schokolade"};
 protected String[] answer = {"Prima!", "Bravo!", "Sehr schön!"};

 protected boolean isSweet = false; // zeigt, welche Liste angezeigt wird

 public void init() {
 // Fenster mit ein paar Bedienungselementen
 Container p = getContentPane();
 p.setLayout(new FlowLayout());

 title = new JLabel("Such was aus:");
 p.add(title);

 selection = new JComboBox(courses);
 selection.addActionListener(new MySelectionListener());
 p.add(selection);

 JButton button = new JButton("Drück mich!");
 button.addActionListener(new MyButtonListener());
 p.add(button);
 }

 protected void switchList() {
 // tauscht Liste der ComboBox aus

 // Bestimmung der neuen Liste
 String[] newList;
 if (isSweet) {
 newList = courses;
 isSweet = false;
 } else {
 newList = sweets;
 isSweet = true;
 }

 selection.removeAllItems(); // löscht alle alten Einträge
 for (int i = 0; i < newList.length; i++) {
 selection.addItem(newList[i]);
 }
 }

 class MyButtonListener implements ActionListener {
 public void actionPerformed(ActionEvent e) {
 // tauscht Liste der ComboBox aus
 switchList();
 }
 }

 class MySelectionListener implements ActionListener {
```

```
public void actionPerformed(ActionEvent e) {
 // ändert das Label je nach Auswahl
 int nr = selection.getSelectedIndex();
 if (nr != -1) { // geschieht beim Wechsel der Liste!
 title.setText(answer[nr]);
 }
}
}
```

# GridLayoutApplet.java

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

public class GridLayoutApplet extends JApplet {

 protected JLabel title;
 protected JComboBox selection;

 protected String[] courses = {"Physik", "Informatik", "Thermodynamik"};
 protected String[] sweets = {"Eis", "Pudding", "Schokolade"};
 protected String[] answer = {"Prima!", "Bravo!", "Sehr schön!"};

 protected boolean isSweet = false; // zeigt, welche Liste angezeigt wird

 public void init() {
 // Fenster mit ein paar Bedienungselementen
 Container p = getContentPane();
 p.setLayout(new GridLayout(3, 1));

 title = new JLabel("Such was aus:", JLabel.CENTER);
 p.add(title);

 selection = new JComboBox(courses);
 selection.addActionListener(new MySelectionListener());
 p.add(selection);

 JButton button = new JButton("Drück mich!");
 button.addActionListener(new MyButtonListener());
 p.add(button);
 }

 protected void switchList() {
 // tauscht Liste der ComboBox aus

 // Bestimmung der neuen Liste
 String[] newList;
 if (isSweet) {
 newList = courses;
 isSweet = false;
 } else {
 newList = sweets;
 isSweet = true;
 }

 selection.removeAllItems(); // löscht alle alten Einträge
 for (int i = 0; i < newList.length; i++) {
 selection.addItem(newList[i]);
 }
 }

 class MyButtonListener implements ActionListener {
 public void actionPerformed(ActionEvent e) {
 // tauscht Liste der ComboBox aus
 switchList();
 }
 }

 class MySelectionListener implements ActionListener {
```



```
public void actionPerformed(ActionEvent e) {
 // ändert das Label je nach Auswahl
 int nr = selection.getSelectedIndex();
 if (nr != -1) { // geschieht beim Wechsel der Liste!
 title.setText(answer[nr]);
 }
}
}
```

# BorderLayoutApplet.java

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

public class BorderLayoutApplet extends JApplet {

 protected JLabel title;
 protected JComboBox selection;

 protected String[] courses = {"Physik", "Informatik", "Thermodynamik"};
 protected String[] sweets = {"Eis", "Pudding", "Schokolade"};
 protected String[] answer = {"Prima!", "Bravo!", "Sehr schön!"};

 protected boolean isSweet = false; // zeigt, welche Liste angezeigt wird

 public void init() {
 // Fenster mit ein paar Bedienungselementen
 Container p = getContentPane();
 p.setLayout(new BorderLayout());

 title = new JLabel("Such was aus:", JLabel.CENTER);
 p.add(title, BorderLayout.NORTH);

 selection = new JComboBox(courses);
 selection.addActionListener(new MySelectionListener());
 p.add(selection, BorderLayout.CENTER);

 JButton button = new JButton("Drück mich!");
 button.addActionListener(new MyButtonListener());
 p.add(button, BorderLayout.SOUTH);
 }

 protected void switchList() {
 // tauscht Liste der ComboBox aus

 // Bestimmung der neuen Liste
 String[] newList;
 if (isSweet) {
 newList = courses;
 isSweet = false;
 } else {
 newList = sweets;
 isSweet = true;
 }

 selection.removeAllItems(); // löscht alle alten Einträge
 for (int i = 0; i < newList.length; i++) {
 selection.addItem(newList[i]);
 }
 }

 class MyButtonListener implements ActionListener {
 public void actionPerformed(ActionEvent e) {
 // tauscht Liste der ComboBox aus
 switchList();
 }
 }

 class MySelectionListener implements ActionListener {
```

```
public void actionPerformed(ActionEvent e) {
 // ändert das Label je nach Auswahl
 int nr = selection.getSelectedIndex();
 if (nr != -1) { // geschieht beim Wechsel der Liste!
 title.setText(answer[nr]);
 }
}
}
```

# BorderLayoutWithBorderApplet.java

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

public class BorderLayoutWithBorderApplet extends JApplet {

 protected JLabel title;
 protected JComboBox selection;

 protected String[] courses = {"Physik", "Informatik", "Thermodynamik"};
 protected String[] sweets = {"Eis", "Pudding", "Schokolade"};
 protected String[] answer = {"Prima!", "Bravo!", "Sehr schön!"};

 protected boolean isSweet = false; // zeigt, welche Liste angezeigt wird

 public void init() {
 // Fenster mit ein paar Bedienungselementen
 Container p = getContentPane();
 p.setLayout(new BorderLayout());

 title = new JLabel("Such was aus:", JLabel.CENTER);
 title.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
 p.add(title, BorderLayout.NORTH);

 selection = new JComboBox(courses);
 selection.addActionListener(new MySelectionListener());
 p.add(selection, BorderLayout.CENTER);

 JButton button = new JButton("Drück mich!");
 button.addActionListener(new MyButtonListener());
 p.add(button, BorderLayout.SOUTH);
 }

 protected void switchList() {
 // tauscht Liste der ComboBox aus

 // Bestimmung der neuen Liste
 String[] newList;
 if (isSweet) {
 newList = courses;
 isSweet = false;
 } else {
 newList = sweets;
 isSweet = true;
 }

 selection.removeAllItems(); // löscht alle alten Einträge
 for (int i = 0; i < newList.length; i++) {
 selection.addItem(newList[i]);
 }
 }

 class MyButtonListener implements ActionListener {
 public void actionPerformed(ActionEvent e) {
 // tauscht Liste der ComboBox aus
 switchList();
 }
 }
}
```

```
class MySelectionListener implements ActionListener {
 public void actionPerformed(ActionEvent e) {
 // ändert das Label je nach Auswahl
 int nr = selection.getSelectedIndex();
 if (nr != -1) { // geschieht beim Wechsel der Liste!
 title.setText(answer[nr]);
 }
 }
}
```

# NiceLayoutApplet.java

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

public class NiceLayoutApplet extends JApplet {

 protected JLabel title;
 protected JComboBox selection;

 protected String[] courses = {"Physik", "Informatik", "Thermodynamik"};
 protected String[] sweets = {"Eis", "Pudding", "Schokolade"};
 protected String[] answer = {"Prima!", "Bravo!", "Sehr schön!"};

 protected boolean isSweet = false; // zeigt, welche Liste angezeigt wird

 public void init() {
 // Fenster mit ein paar Bedienungselementen
 Container p = getContentPane();
 p.setLayout(new BorderLayout());

 // Bereich mit FlowLayout für Label und Button
 JPanel upperPanel = new JPanel(new BorderLayout());
 upperPanel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
 p.add(upperPanel, BorderLayout.NORTH);

 title = new JLabel("Such was aus:", JLabel.CENTER);
 title.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
 upperPanel.add(title, BorderLayout.WEST); // ins upperPanel

 selection = new JComboBox(courses);
 selection.addActionListener(new MySelectionListener());
 p.add(selection, BorderLayout.CENTER);

 JButton button = new JButton("Drück mich!");
 button.addActionListener(new MyButtonListener());
 upperPanel.add(button, BorderLayout.EAST); // ins upperPanel
 }

 protected void switchList() {
 // tauscht Liste der ComboBox aus

 // Bestimmung der neuen Liste
 String[] newList;
 if (isSweet) {
 newList = courses;
 isSweet = false;
 } else {
 newList = sweets;
 isSweet = true;
 }

 selection.removeAllItems(); // löscht alle alten Einträge
 for (int i = 0; i < newList.length; i++) {
 selection.addItem(newList[i]);
 }
 }

 class MyButtonListener implements ActionListener {
 public void actionPerformed(ActionEvent e) {
```

```
 // tauscht Liste der ComboBox aus
 switchList();
 }
}

class MySelectionListener implements ActionListener {
 public void actionPerformed(ActionEvent e) {
 // ändert das Label je nach Auswahl
 int nr = selection.getSelectedIndex();
 if (nr != -1) { // geschieht beim Wechsel der Liste!
 title.setText(answer[nr]);
 }
 }
}
}
```

# Applets

- Primzahl
- HalloApplet
- ZeichenApplet
- EventApplet
- GridLayoutApplet
- BorderLayoutApplet
- BorderLayoutWithBorderApplet
- NiceLayoutApplet