Grundlagen

Java-Programme werden durch eine Compiler in einen Zwischencode umgesetzt, der dann nach dem Programmaufruf von einer „Laufzeitumgebung“ interpretiert wird. Das Verfahren hat den Vorteil, dass Programme auf vielen Umgebungen (Prozessoren, Betriebssystemen) laufen, ohne angepasst zu werden (Ausnahme: grafische Oberfläche) . Man benötigt also einen Compiler und die zum Rechner passende Laufzeitumgebung. Beim Erstellen eines Programmes sollte man sich an dem Programmschema orientieren, wie er unten im Abschnitt „Java-Aufbau“ beschrieben ist. Dies dient der eigenen Übersicht!
Java ist übrigens Ausgangspunkt von mehreren aktuellen Sprachen!

Begriffe:

Klassen sind Zusammenfassungen von Methoden und Variablendeklarationen.
Deklarationen organisieren Objekte und benennen diese
Variablendeklarationen organisieren Speicherplätze (Variable) und benennen diese.
Methoden fassen Aktionen zusammen.

Beispiel 1 (Adressen - Konsole):

```
import java.io.*;                                   Einbinden zusätzlicher Befehle für Konsolenprogramme

public class adrneu                                 Beginn der Hauptklasse, die nach dem Kompilieren aufgerufen wird, wichtig: Dateiname = Klassenname!
 {
  public static void main (String args[]) {        Hauptmethode, wird bei Programmstart aufgerufen
    adr alt[]=new adr[3];                           Variablendeklaration
    for (int i=0;i<=2;i++) {                        Schleife:
      alt[i]=new adr();                               Erzeugen von neuen Adressen
      alt[i].in();};                                  Eingabe der Adressdaten
    for (int i=0;i<=2;i++) {                        Schleife:
      alt[i].out();}                                  Ausgabe der Adressen
   }                                                Methodenende
 }                                                  Klassenende

class adr                                           Beginn der (Hilfs-)Klasse adr
 {
  byte[] name=new byte[80];                         Variablendeklaration für alle Methoden in adr

  public void in() {                                Methode zur Dateneingabe
    int zahl=0;                                     Variablendeklaration
    System.out.println("Adresse :");                Anzeige von „Adresse:“
    try {zahl=System.in.read(this.name);}           Dateneingabe
    catch (IOException e) {System.out.print("Fehler");};   Fehlerbehandlung bei Eingabeproblemen
    };                                              Methodenende

  public void out() {                               Methode zur Datenausgabe
    System.out.println("Adresse :");                Anzeige von „Adresse:“
    System.out.write(this.name,0,80);               Anzeige von Daten
 } }                                                Methodenende, Klassenende
```

Beispiel 2 (einarmiger Bandit - Applet):

```
import java.awt.*, java.applet.*;                   Einbinden zusätzlicher Befehle für Applets

public class bandit2 extends Applet {               Hauptklasse
  Button b1=new Button("  1  "),                    Deklaration dreier Button
  b2=new Button("  2  "),
  b3=new Button("  3  ");
  Label l1=new Label("000"),                        Deklaration dreier Anzeigefelder
  l2=new Label("000"),
  l3=new Label("000");
  int d1=1,d2=1,d3=1;                               Deklaration von Zahlenvariablen, Wert jeweils 1

  public void init() {                              Startmethode
    add(l1);add(l2);add(l3); add(b1);add(b2);add(b3);   Anordnen von Button und Anzeigefeldern auf dem Bildschirm
  }                                                 Methodenende
```

```
 public void paint(Graphics g){                  Methode zum Bildaufbau
   String te="";                                 Deklaration von Textvariablen te
   int za=0;                                     Deklaration von Zahlenvariablen za
   te=l1.getText();                              Auslesen von Anzeigefeld in te
   za=new Integer(te).intValue()+d1;             Umwandeln von Text te in Zahl, d1 addieren, Speichern in za
   l1.setText(""+za);                            Überschreiben von Anzeigefeld mit za
   te=l2.getText();                              dgl. für 2. Anzeigefeld
   za=new Integer(te).intValue()+d2;
   l2.setText(""+za);
   te=l3.getText();                              dgl. für 3. Anzeigefeld
   za=new Integer(te).intValue()+d3;
   l3.setText(""+za);
   try{Thread.sleep(1000);}                      Zeitverzögerung
   catch(Exception e){};
   repaint();                                    Anzeige der neuen Inhalte
 }                                               Methodenende

 public boolean action(Event e,Object o){        Methode, die auf Button reagiert
   if (e.target==b1) d1=Math.abs(d1-1);          wenn 1.Button gedrückt, d1 auf 0 setzen
   if (e.target==b2) d2=Math.abs(d2-1);          dgl. 2. Button
   if (e.target==b3) d3=Math.abs(d3-1);          dgl. 3. Button
   return true;                                  Rückgabe „true“, von Windows erfordert (keine Fehlermeldung)
} }                                              Methodenende, Klassenende
```

Java-Aufbau

Konsolenprogramme:

```
import.java.io.*;

public class haupt                          Klassen- = Dateiname (hier: haupt)!!
{
  Variablendeklarationen
  Methodendeklarationen

  public static void main(String du[])
  {
    Befehle
  }
}
```

zusätzliche Klassen (wie Hauptklasse ohne main)

Variablendeklaration:

*Typ (*boolean, int, double, char, String, *selbstdefinierte Klasse) Variablenname = Anfangswert;*

Methodendeklaration:

```
public static Variablentyp (oder void) Methodenname (Parameterliste)
                                            public, static nur bei Bedarf
{                                           (public: überall verwendbar,
  Variablendeklarationen                     static: ohne Objektbezug verwendbar)
  Befehl1;
  Befehl2;
  :
  return Term                               Rückgabe von Termwert, nicht bei void
}

public Klassenname(Parameterliste)          Konstruktor: Erzeugen eines neuen Objekts
```

{
  Befehle
}

Rückgabe: Adresse des Objekts

Parameterliste:

Typ1 Name1 , Typ2 Name2 ...

Die Namen können beim Aufrufen der Methode durch Variablennamen oder Terme ersetzt werden

Befehle:

Methodenaufruf

if(*Bedingung*) *Befehl1*;
else *Befehl2*;

wenn Bedingung erfüllt, dann *Befehl1*, sonst *Befehl2*

while(*Bedingung*) *Befehl;*

solange Bedingung erfüllt, *Befehl*

for(*Initialisierung*;*Bedingung*;*Befehl1*) *Befehl2*;

Initialisierungsbefehl, solange Bedingung erfüllt, *Befehl2*,*Befehl1*

Regeln:

1) Jeder Befehl ist mit ; abzuschließen, Ausnahme: } .
2) Mehrere Befehle können durch { } zu einem Befehl geklammert werden.

Bedingung:

Term mit Abfragen (&& und, || oder , ! nicht)

Abfrage:

Term == *Term*, *Term* != *Term*, *Term* <= *Term*, *Term* < *Term*, *Term* >= *Term*, *Term* > *Term*
String.equals(*Text*)
String.compareTo(*Text*) (Ergebnisse: -1,0,1)

spezielle Befehle:

String.charAt(*Position*)

Zeichen an Postion in *String* (Typ char)

String.indexOf(*Text*)

Position von Text in *String*

String.substring(*Anf*,*End*)

Teilstring mit Anfang *Anf*, Ende *End* (exkl.)

String1+*String2*

Anfügen von *String2* an *String 1*

new Integer(*String*).intValue()

Umwandlung Text in Zahl (dgl. float usw.)

(*Typ*) *Term*

Typ vom Termergebnis

System.out.print(*Text*)

Bildschirmausgabe von *Text*

DataInputStream *Name*=new DataInputStream(System.in)

Einrichten von Tastaturpuffer

Name.readLine()

Lesen aus dem Tastaturpuffer

try

Versuch, nachfolgende Befehle auszuführen

{
  Befehle
}
catch(Exception *Name*)

wenn Fehler, dann nachfolgende Befehle

{
  Befehle

}

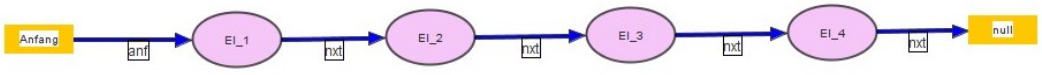
Datenstrukturen:

Typ Name[] = new *Typ*[*anzahl1*][*anzahl2*] — Array mit *anzahl1* Zeilen, *anzahl2* Spalten
Name[*2*][*4*] — Speicher Zeile Nr. 2, Spalte Nr. 4

class liste — Listenelement

```
{
  Datendeklarationen
  liste nxt=null;                                  Zeiger auf Nachfolger (Speicheradresse)

  public liste(Parameterliste, liste altanf)       Erzeugen eines neuen Elements am
  {                                                Listenanfang
    Befehle
    nxt=altanf;                                    Nachfolger ist der alte Listenanfang
  }                                                Rückgabe: Zeiger auf neuen Listenanfang
  public liste suchen(bedingung)                   Element mit bedingung suchen
  {
    liste erg=null;                                Speicher erg deklarieren, erg=null
    if(bedingung) erg=this;                        wenn bedingung erfüllt, dann in  erg Zeiger
                                                   auf aktuelles Element speichern
    else if(nxt!=null) erg=nxt.suchen(bedingung);  sonst Nachfolger untersuchen
    return erg;                                    Rückgabe: Zeiger auf Element mit bedingung
  }
}
```



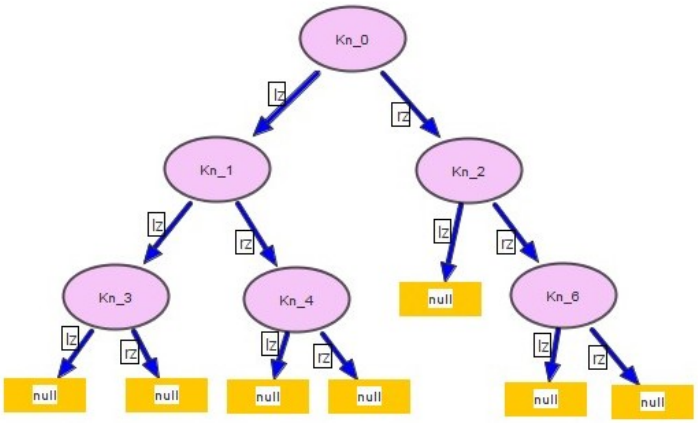
class baum — Baumknoten

```
{
  Datendeklarationen
  baum lz=null,rz=null;                            Zeiger auf linken und rechten Nachfolger

  public baum(Parameterliste, baum li,baum re)     Erzeugen eines neuen Knoten
  {
    Befehle
    lz=li;rz=re;                                   linker Nachfolger ist li. rechter ist re
  }                                                Rückgabe: Zeiger auf neuen Knoten

  public baum suchen(term,wert)                    Knoten mit term = wert suchen
  {
   baum erg=null;                                  Speicher erg deklarieren, erg=null
   if(term==wert) erg=this;                        wenn bedingung erfüllt, dann in  erg Zeiger
                                                   auf aktuelles Element speichern
   else
   {
    if(term<wert && lz!=null) erg=lz.suchen(term,wert);      linken Nachfolger untersuchen
    else
      if(term>wert && rz!=null) erg=rz.suchen(term,wert);    rechten Nachfolger untersuchen
   }
   return erg;                                     Rückgabe: Zeiger auf Element mit term=wert
```

```
  }
}
```



Applets (Grafik auf Internetseiten) :

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

public class haupt extends Applet                    Klassen- = Dateiname (hier: haupt)!!
{
  public void init()                                 Startmethode, u.a zur Definition von Ein-
                                                     gabeelementen wie Button
  public void paint(Graphics g)                      Grafikausgabe mit Grafikobjekt g (Standard-
                                                     bildschirm oder virtueller Bildschirm)
  public void update(Graphics g)                     wie paint(), jedoch ohne Bildschirmlöschen

  public boolean handleEvent(Event e)                Reaktion auf Maus-/Tastatur-Betätigung
                                                     (Windows-Meldung in e übergeben)
  public boolean action(Event e,Object o)            Reaktion auf Anklicken mit der Maus
  public boolean keyDown(Event e,int k)              Reaktion auf Betätigen der Tastatur
                                                     (Tastaturkode von Windows in k übergeben)
}
```

Event e:

```
e.x, e.y                                             Mauskoordinaten
e.id                                                 Art der Mausaktion (Zahlenwert!)
e.target                                             angeklicktes Objekt
e.key                                                Kode der betätigten Taste
```

Grafikobjekte:

```
Panel()                                              Anzeigefläche
Button(Text)                                         Button mit Aufschrift
Label(Text)                                          Textausgabe
TextField(Text)                                      Eingabezeile mit Textvorgabe
```

Grafikgestaltung:

```
repaint()                                            Neuerstellen der Grafik
```

g: von Windows übergebenes oder selbst erzeugtes Grafikobjekt:

new Color(*Rot*,*Grün*,*Blau*)	neue Zeichenfarbe aus Farbanteilen
setBackground(*Farbe*)	Aktivieren der Hintergrundfarbe
g.setColor(*Farbe*)	Aktivieren von *Farbe*
g.drawRect(x,y,*Breite*, *Höhe*)	Rechteckrand (x,y obere linke Ecke)
g.fillRect(x,y,*Breite*,*Höhe*)	ausgefüllte Rechteck
g.drawOval(x,y,*Breite*,*Höhe*)	Ellipsenrand
g.fillOval(x,y,*Breite*,*Höhe*)	ausgefüllte Ellipse
g.drawLine(xa,ya,xe,ye)	Strecke von (xa,ya) bis (xe,ye)
g.drawPolygon(x[],y[],n)	n-Eck-Rand mit den Eckpunkten (x[0],y[0]), (x[1],y[1]), ...
g.fillPolygon(x[],y[],n)	ausgefülltes n-Eck
new Font(*Fontname*,*Gestalt*,*Größe*)	neuer Font mit Gestaltart Font.BOLD, Font.ITALIC oder Font.PLAIN
g.setFont(*Font*)	Aktivieren von *Font*
g.drawString(*Text*,x,y)	Textausgabe ab Position (x,y)

Frames (eigenständige Grafik):

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

public class haupt extends Frame                Klassen- = Dateiname (hier: haupt)!!
{
  Grafik-Methoden wie für Applets

  public static void main(String du[])          Startroutine, von der die restlichen Methoden
                                                gestartet werden (unbedingt erforderlich!)
}
```

Hinweis: Dies ist nur ein kleiner Ausschnitt der Möglichkeiten für die Grafikgestaltung!

Threads (parallele Prozesse):

```
Thread th=new Thread(kla)                       Deklaration eines Prozesses, der mit der
                                                Klasse kla arbeitet (siehe unten).
th.start()
th.stop()
th.wait()                                       Stoppt den Prozess, bis er von anderen mit
  th.notify()                                     wieder gestartet wird
th.sleep(Millisek)                              Friert den Prozess th Millisek lang ein
Thread.sleep(Millisek)                          Friert den aktuellen Prozess ein

public class kla ... implements Runnable        Deklaration der Prozessklasse kla
{
  public void run()                             Startroutine (unbedingt erforderlich!)
}
```

Dateien:

```
String te;
FileInputStream fi=new FileInputStream(Name);   Lesepuffer zur Datei Name einrichten
DataInputStream di=new DataInputStream(fi);
if(di.available()>0) te=di.readLine();          Wenn noch nicht Dateiende erreicht, Text-
```

	zeile aus di in te einlesen
di.close();	Schließen der Datei, Ende des Lesens
FileOutputStream fo=new FileOutputStream(*Name*);	Schreibpuffer zur Datei *Name* einrichten
DataOutputStream do=new DataOutputStream(fo);	
do.writeChars(te);	Schreiben des Textes te in do
do.close();	Übertragen der restlichen Daten aus do in die Datei, Schließen der Datei

Hinweis: Fehlerbehandlung mit try.. catch.. erforderlich!

Math. Funktionen (Auswahl)**:**

+,-,*/,%	Addition, Subtraktion, Multiplikation, Division, Divisionsrest
Math.pow(x,y)	Potenz x^y
Math.sin(x), Math.cos(x), ...	trigonometrische Funktionen
Math.min(x,y), Math.max(x,y)	kleinere bzw. größere der Zahlen x und y