Grundbegriffe

der objektorientierten Programmiersprache Java lassen sich anschaulich am Beispiel einer (virtuellen) Automobilmesse verdeutlichen:
Die Klassen (Automarken: Volvo, Daimler-Chrysler, VW, Toyota, Ford, Opel, ...) sind in einzelnen Paketen zusammengefasst, welche z. B. den Themenschwerpunkten LKW, PKW, Sportwagen entsprechen.
Unter Objekten versteht man die konkreten Modelle der verschiedenen Produzenten; Parameter (Instanzvariablen) wie Typ, Motorisierung, Farbe, Ausstattung charakterisieren jedes spezifische Automobil (Instanz).
Das Zusammenspiel technischer Komponenten (z. B. Motor, Getriebe, Elektrik, Elektronik) eines Autos kann man mit den Methoden in Java vergleichen.
Abstrakte Begriffe wie Antrieb oder Steuerung stellen untergeordnete Subklassen dar, die erst durch Konstruktoren (z.B. Konstruktionsunterlagen) zum realen Objekt werden.
Entwickler, Ausrüster oder Zulieferer (Programmierer) können abstrakte Entwürfe (Klassen) oder reale Umsetzungen (Objekte, Methoden) neuer bzw. weiterentwickelter Komponenten als Zubehör (z. B. Audioanlage, Klimaautomatik, ...) anbieten, welche das originale Fahrzeug (Superklasse) ergänzen bzw. verändern.
Allen Autos gemeinsam, unabhängig von Hersteller oder Typ, sind Ausrüstungen wie z. B. ABS, TRC, ESP, ACC, ... vorausgesetzt sie wurden über die Zubehörliste geordert (implementiert). Fahrzustände (Bremsen, Beschleunigen, ...) aktivieren diese Komponenten. In Java werden solche Ergänzungen, die unterschiedlichen Klassen zur Verfügung gestellt werden können, Schnittstellen genannt. Die Steuerung der Schnittstellen mittels Tastatur oder Maus überwachen sogenannte Listener, welche mit Sensoren und Mikroprozessoren im Auto vergleichbar sind.

Klasse

abstrakte Vorlage für ein Objekt mit:

Attributen (Eigenschaften des Objektes),
Methoden (Aktivitäten, Verhalten des Objektes)

z.B.

- Klasse Graphics:
  Umgebung zum Zeichnen von Grafik und Text
- Klasse Color:
  Farbskala
- Klasse Math:
  mathematische Operationen

Jedes Programm in Java besteht aus mindestens einer Klasse. Klassen dienen als Schablone zur Erzeugung von Objekten. Attribute einer Klasse werden über Variablen definiert, Operationen einer Klasse dagegen durch ihre Methoden deklariert. Die Variablen einer Klasse beschreiben den Zustand eines Objektes, die Methoden charakterisieren dessen Verhalten. Klassen können hierarchisch geordnet sein, d. h. übergeordnete Klassen (Superklassen) übertragen (vererben) ihre Attribute und Verhaltensweisen auf untergeordnete Klassen (Subklassen).
Jede Klasse wird mit dem Schlüsselwort class eingeleitet.
Klassennamen beginnen mit einem Großbuchstaben; ein neuer Wortstamm innerhalb des Klassennamens wird zur besseren Lesbarkeit wieder mit einem Großbuchstaben eingeleitet.
Der Klassenname legt den Dateinamen fest, unter dem die Klasse abzuspeichern ist.
Die folgenden Namen, Sprachelemente von Java, sind reserviert und stehen nicht zur Auswahl:
abstract, asset, boolean, break, byte, byvalue, case, cast, catch, char, class, const, continue, default, do, double, else, extends, false, final, finally, float, for, future, generic, goto, if, implements, import, inner, instanceof, int, interface, long, native, new, null, operator, outer, package, private, protected, public, rest, return, short, static, super, switch, synchronized, this, throw, throws, transient, true, try, var, void, volatile, while.

Pakete

Sammlung (Gruppierung) von Klassen

- Standardmäßig hat jeder Programmierer Zugriff auf das Paket java.lang welches die Sprachkomponenten von Java enthält.
- Allgemeine Basis-Routinen umfasst das Paket java.util.
- Mit dem Paket java.awt (Abstract Windowing Toolkit) lassen sich grafische Benutzeroberflächen (GUI) erzeugen; es stellt entsprechende Komponenten wie Schaltflächen, Beschriftungen und Textfelder zur Verfügung. Bestandteile dieses Pakets sind z. B. die bereits erwähnten Klassen Graphics bzw. Color. Um eine Klasse aus diesem Paket (AWT) zu importieren, ist der Bezug auf das Paket in Punktnotation anzugeben; das zugehörige Schlüsselwort ist import.

  ```
  import java.awt.*
  ```

- Klassen zur Erzeugung grafischer Komponenten (Buttons, Werkzeugleisten, Bildlaufleisten, Dialogfenster, Regler, ...) sind im Paket javax.swing, einer Weiterentwicklung des Abstract Window Toolkit, enthalten.
- Klassen zur Ein- und Ausgabe von Daten stellt das Paket java.io bereit.
- java.net beinhaltet alle Klassen zum Thema Netzwerk.

Objekt

Instanz (konkrete Realisierung) einer Klasse mit:

Zugriff auf alle Methoden der zugehörigen Klasse,
konkreten Werten der (Instanz)-Variablen

z.B.

- Farbobjekt:
  Color(0,128,255)
  konkreter Farbwert definiert
  über die Instanzvariablen der Rot-, Grün- und Blau-Anteile
- PrintStream-Objekt System.out.println(...)
  Standardausgabe (zeilenweise) des Computers

In der objektorientierten Programmierung (OOP) stehen Objekte für separate Teile des Programms, die interagieren um vorgesehene Aufgaben zu erledigen. Objekte weisen Attribute auf. Objekte inklusive ihrer Variablen beanspruchen Speicherplatz, der von Java automatisch verwaltet wird.

Objekte werden mit dem Operator new gefolgt vom Namen der Klasse erzeugt, von der Sie eine Instanz anlegen wollen.

```
Klassenname objektname = new Klassenname(...)
```

In Klammern stehen die Anfangswerte der zugehörigen Argumente;
z.B.

```
Point location = new Point(0, 0);
```

Die Klammern können auch leer bleiben; in diesem Fall wird ein einfaches Objekt generiert.
Das Schlüsselwort new

- ruft den Konstruktor (s. u.) der Klasse auf
- weist dem Objekt Speicherplatz zu
- legt die Anfangswerte der Variablen fest
- erstellt das Objekt

Objekte werden im Programm über die zugehörigen Referenzvariablen in Form des Objektnamens aufgerufen.

Methode (Funktion)

Anweisungen (aktive Elemente) innerhalb einer Klasse zu:

Verhaltensweisen aller Objekte dieser Klasse
(vergleichbar mit Funktionen in anderen Programmiersprachen)
Die Argumente einer Methode stehen in Klammern, welche dem Aufruf der Methode folgen müssen. Hat die Methode keine Argumente, so sind leere Klammern ( ) zu schreiben, z. B.

```
e.toString( )
schieber.getValue( )
repaint( )
```

Ausgewählte Methoden der Klasse java.awt.Graphics sind:

- Zeichnen von Zeichenketten, Umrissen
  drawString(...), drawLine(...), drawRect(...), drawOval(...)
- Zeichnen gefüllter Formen
  fillRect(...), fillOval(...)
- Festlegen von Farben
  setColor(...)
- Festlegen von Schriftarten
  setFont(...)

Objekte können nur über Methoden miteinander kommunizieren. Methoden enthalten den ausführbaren Code einer Klasse. Sie werden aus Anweisungen gebildet, abgeschlossen mit einem Semikolon (;). Jede Anweisung sollte aus Gründen der Überschaubarkeit in einer eigenen Zeile stehen. Anweisungen werden in Blöcken innerhalb geschweifter Klammern { } zusammengefasst.
Methoden können Objekte erzeugen, Ausdrücke auswerten oder andere Methoden aufrufen. Methodennamen beginnen mit kleinen Buchstaben. Vor dem Methodennamen stehen durch Leerzeichen voneinander getrennt Deklarationen.
z.B.

- public (öffentlich)
  Jedes fremde Objekt darf auf die Methode zugreifen.
- static
  Diese (globale) Methode gehört zur Klasse und ist nicht nur einzelnen Objekten der Klasse zugeordnet.
- final
  Variablen besitzen einen konstanten, unveränderlichen Wert.
- void
  Die Methode liefert keinen Wert zurück.
- private
  Der Zugriff wird auf das Objekt selbst beschränkt; fremde Objekte können auf diese Teile nicht zugreifen.
- protected
  Zugriff von allen Objekten, deren Klassen aus der aktuellen Klasse abgeleitet sind

Das Ergebnis einer Methode, liefert die Anweisung return, welche die Methode beendet und die Übergabe des Resultats an das übergeordnete Programm veranlasst; ausgenommen ist nur die Deklaration void.

Eine Klasse kann mehrere Methoden mit demselben Namen verwalten (Überladen von Methoden); sie unterscheiden sich durch die Parameterliste, d.h. durch Anzahl und Datentyp der Parameter.
Die Methode zur Erstellung von Objekten einer Klasse nennt man Konstruktor. Der Konstruktor kann eine von mehreren Methoden ohne Rückgabewert sein und ist Bestandteil jeder Klasse. Konstruktoren haben denselben Namen wie die Klasse, zu der sie gehören. Im Allgemeinen weist ein Konstruktor Objekten Anfangswerte zu, stellt Speicherplatz bereit und generiert damit eine konkrete Instanz des Objektes. Ohne Konstruktor können keine Objekte erzeugt werden. Jede Klasse, die nicht über eigenen Konstruktor verfügt, erhält durch den Compiler automatisch den Konstruktor einer übergeordneten Klasse zugewiesen.
Konstruktoren können wie jede Methode überladen werden; verschiedene Konstruktoren in einer Klasse unterscheiden sich durch den Typ und/oder die Anzahl ihrer Argumente. Der Java-Compiler wählt automatisch den zur vorgegebenen Parameterliste gehörigen Konstruktor aus.
z.B.

Die Klasse java.lang.String hat u.a. Konstruktoren mit folgenden Parameterlisten:

- ( )
  public java.lang.String ( )
- (byte[ ])
  public java.lang.String (byte[ ])
- (byte[ ], int)
  public java.lang.String (byte[ ], int)
- (byte[ ], int, int)
  public java.lang.String (byte[ ], int, int)
- (char[ ])
  public java.lang.String (char[ ])
- (char[ ], int, int)
  public java.lang.String (char[ ], int, int)

Beachten Sie die Punktnotation in der obigen Schreibweise; links vom Punkt wird die Zugehörigkeit des String-Objektes (Zeichenkette) zum Paket java.lang verdeutlicht.
Die einzelnen Konstruktoren unterscheiden sich durch den Datentyp zur Speicherung einer Zeichenkette in Form von Bytes oder 16-Bit-Unicode-Zeichen sowie durch Anzahl der Literale oder Segmente aus der Zeichenkette.
Da Methoden i. Allg. auf konkrete Objekte wirken, werden diese Beziehungen im Quelltext eines Programms ebenfalls mit der Punktnotation vorgegeben; links vom Punkt steht dann das Objekt, auf welches die Methode wirkt. Den Methodennamen ergänzt eine Parameterliste in runden Klammern; auch bei spezifischen Instanzvariablen klärt die Punktnotation eindeutig Relationen zwischen Parameter und Klasse.
z.B.

```
g.setColor(Color.MAGENTA)
Scrollbar(Scrollbar.VERTICAL, 51, 5, 1, 101)
```

Der Transfer der Parameter (Argumente) an eine Methode erfolgt in der Reihenfolge von links nach rechts, wobei die Liste die genaue Anzahl und den genauen Typ der Parameter, entsprechend den Vorgaben der Methode, einhalten muss.
Am Beispiel der Klasse java.lang.Math, die alle mathematischen Standardfunktionen zusammenfasst, die im Java-Standardpaket enthalten sind, soll der Terminus Methode noch einmal verdeutlicht werden.

```
public final synchronized class

    java.lang.Math

    extends java.lang.Object

{
```

Methoden:

```
    public static double abs (double)

    public static float abs (float)

    public static int abs (int)

    public static long abs (long)

    public static double acos (double)

    public static double asin (double)

    public static double atan (double)

    public static double atan2 (double, double)

    public static double cbrt (double)          kubische Wurzel

    public static double ceil (double)          Aufrundung auf nächste Ganzzahl

    public static double cos (double)

    public static double cosh (double)

    public static double exp (double)

    public static double expm1 (double)         exp(x) - 1

    public static double floor (double)         Abrundung auf nächste Ganzzahl

    public static double IEEEremainder (double, double)
```

```
    public static double log (double)
    public static double log10 (double)
    public static double log1p (double)          ln(x+1)
    public static double max (double, double)
    public static float max (float, float)
    public static int max (int, int)
    public static long max (long, long)
    public static double min (double, double)
    public static float min(float, float)
    public static int min (int, int)
    public static long min (long, long)
    public static double pow (double, double)
    public static double random ( )
    public static double rint (double)           Rundung auf nächste Ganzzahl
    public static long round (double)
    public static int round (float)
    public static double signum (double)
    public static double sin (double)
    public static double sinh (double)
    public static double sqrt (double)
    public static double tan (double)
    public static double tanh (double)
    public static double toDegrees (double)
    public static double toRadians (double)
}
```

Seit Java 5 ist es möglich, mathematische Methoden oder Konstanten mit static import zu importieren und alle Elemente der Klasse Math ohne das entsprechende Präfix zu verwenden. Beachten Sie: Einige der Methoden, wie z. B. Hyperbolische Funktionen, stehen dem Programmierer erst seit der Entwicklungsversion Java 2 Standard Edition 5.0 (J2SE 5.0) zur Verfügung.

Schnittstelle (interface)
Beschreibung abstrakter Verhaltensweisen, die von allen Klassen unterstützt werden müssen, welche das Interface implementieren:

Komplexe Programme mit grafischer Benutzeroberfläche (GUI) lassen Aktionen des Anwenders zu, mit denen dieser in den Ablauf des Programms eingreifen kann (Tastatureingaben, Mausklicks, Schaltflächen, Schieberegler, Fenster, Aufklappmenüs usw.). Auf entsprechende Ereignisse (Events) wird im Programm über Schnittstellen, die als EventListener bezeichnet werden, reagiert.

Schnittstellen zur Verarbeitung von Ereignissen sind:

ActionListener
Ereignisse, die durch die Aktionen eines Benutzers (z. B. Klick auf einen Button) ausgelöst werden (JButton, JCheckBox, JComboBox, JRadioButton)
z.B:

```
class RadioButtonListener implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();
    switch(command.charAt(0)) {
      case 'I...':
      ...
      break;
      case 'II...':
      ...
      break;
      case 'III...':
      ...
      break;
    }
  }
}
```

ChangeListener (Paket Swing)
Überwachung von Wertänderungen an Schiebereglern (JSlider),
z.B:

```
class SliderListener implements ChangeListener {
  public void stateChanged(ChangeEvent e) {
    JSlider source = (JSlider)e.getSource();
      // Wertübergabe nicht während der Regelung
      // sondern erst nach deren Beendigung
      if (!source.getValueIsAdjusting()) {
        schieberWert = (int)((JSlider)e.getSource()).getValue();
```

```
            ...
        }
    }
}
```

Im Paket AWT kontrolliert die Schnittstelle AdjustmentListener mit ihrer Methode adjustmentValueChanged() die Klasse Scrollbar (Bildlaufleiste).

FocusListener
: Komponente, z.B. Textfeld, erhält oder verliert den Eingabefokus

KeyListener
: Reaktionen bei der Eingabe einzelner Zeichen über die Tastatur

MouseListener
: Reaktionen auf Aktionen mit der Maus

MouseMotionListener
: Ereignisse infolge von Mausbewegungen über eine Komponente

WindowListener
: Ereignisse bei der Maxi-, Minimierung, Verschiebung oder Schließung von Fenstern (JFrame, JWindow)

Wird eine Schnittstelle mit einer Klasse verknüpft, so muss diese Klasse alle Methoden der Schnittstelle einbauen. Alle Klassen die z. B. ActionListener implementieren, müssen über deren einzige Methode actionPerformed( ) verfügen. Durch das Schlüsselwort implements wird dem Compiler angezeigt, welche Schnittstellen eingefügt werden sollen.
z. B.

```
public class Klasse01 implements Schnittstelle01
```

Variablen:
Speicherplatz zur Verwaltung von Informationen

- Klassenvariablen
  spezifische Eigenschaften einer ganzen Klasse
  z.B.
  Klasse java.lang.System
  mit statischen Konstanten (Fields):
  java.io.PrintStream err
  java.io.InputStream in
  java.io.PrintStream out

  Klasse java.awt.Color
  mit statischen Konstanten (Fields):
  black, blue, cyan, darkGrey, gray, green, lightGray, magenta, orange, pink, red, white, yellow

  Klasse java.lang.Math
  mit statischen Konstanten (Fields):
  Math.E = 2.718281828459045
  Math.PI = 3.1415926535589793

- Instanzvariablen (Objektvariablen)
  Attribute (Erscheinung) eines konkreten Objektes,
  Informationsaustausch zwischen einzelnen Objekten

- Lokale Variablen
  innerhalb von Methoden

Für die Benennung von Variablen hat sich folgende Konvention durchgesetzt:

- Variablennamen beginnen mit einem Kleinbuchstaben.
- Jedes folgende Wort fängt mit einem Großbuchstaben an.
- Die restlichen Buchstaben sind Kleinbuchstaben.

Die Namen von Klassen- oder Instanzvariablen basieren ebenso wie die von Methoden auf einer Punkt-Notation; der Bezug auf eine Klasse oder das konkrete Objekt steht links vom Punkt, die eigentliche Variable und die Methode rechts davon. Variablennamen dürfen nicht mit einer Ziffer beginnen. Namen bestehend aus einem Buchstaben werden üblicherweise nur für:

Zähler (Laufvariablen)	i,j,k...
Koordinaten	x,y,z

verwendet.

Elementare Variablentypen

speichern ganze Zahlen, Gleitkommazahlen, Boolsche Werte oder Zeichenketten:

- boolean — true, false (Logik)
- char — 16-Bit Unicode Standard (Literale)
- byte — ganze Zahl 8 bit (-128...127)
- short — ganze Zahl 16 bit (-32768...32767)
- int — ganze Zahl 32 bit (-2147483648...2147483647)
- long — ganze Zahl 64 bit
- float — Gleitkommazahl 32 bit Genauigkeit 6-7 Stellen)
- double — Gleitkommazahl 64 bit (Genauigkeit 15-16 Stellen)

Array

Sammlung von Variablen gleichen Typs

Ein Array wird dadurch definiert, dass man hinter die Bezeichnung, welche den Typ der zugehörigen Variablen festlegt, eckige Klammern setzt.

z.B.

```
int[ ] values = new int[10]
```

Der Zugriff auf die Komponenten eines Arrays (im vorliegenden Beispiel values) erfolgt über ganzzahlige Indizes (im Beispiel von 0 bis 9).

Vererbung

Gemeinsame Attribute oder Methoden werden innerhalb der objektorientierten Programmierung in übergeordneten Klassen zusammengefasst. Jede untergeordnete Klasse kann Datenfelder und Verhaltensweisen der übergeordneten Klasse übernehmen. Die von der Superklasse ererbten Eigenschaften können um weitere Attribute oder Methoden erweitert werden; Schlüsselwort dafür ist das Verb extends (erweitern).

z. B.

```
public class Klasse02 extends Klasse01
```

Die Zugriff auf eine Methode oder auf eine Variable erfolgt in der ersten, relevanten Klasse oberhalb der aktuellen Klasse, in der diese gefunden werden.

Ererbte Methoden können überschrieben (override) werden, d. h. der Programmierer verändert einzelne Verhaltensweisen von Methoden der übergeordneten Klasse, um sie konkreten Gegebenheiten anzupassen. Dazu definiert man die Methode in der Subklasse exakt so wie in der übergeordneten Klasse (Name, Rückgabetyp, Parameterliste) und ergänzt sie um zusätzliche Spezialisierungen. Ein Beispiel dafür ist die Methode paint( ), die von der Klasse java.awt.Component ererbt wird. Die Methode, die standardmäßig leer ist, wird immer dann aufgerufen, wenn grafische Inhalte (Text, Hintergrund, Bild, ...) eines Fensters neu dargestellt werden müssen.

Die Klasse java.awt.Color verdeutlicht mit ihren Inhalten noch einmal alle bisher definierten Begriffe.

```
public synchronized class

        java.awt.Color

        extends             java.lang.Object

        implements          java.awt.Paint,
                            java.io.Serializable

{
```

Fields:

```
  public static final java.awt.Color black;
  public static final java.awt.Color blue;
  public static final java.awt.Color cyan;
  public static final java.awt.Color darkGray;
  public static final java.awt.Color gray;
  public static final java.awt.Color green;
  public static final java.awt.Color lightGray;
  public static final java.awt.Color magenta;
  public static final java.awt.Color orange;
  public static final java.awt.Color pink;
  public static final java.awt.Color red;
  public static final java.awt.Color white;
  public static final java.awt.Color yellow;
```

Konstruktoren:

```
        ( float, float, float )
                public java.awt.Color (float, float, float)

        ( float, float, float, float )
                public java.awt.Color (float, float, float, float)

        ( int )
                public java.awt.Color (int)

        ( int, int, int )
                public java.awt.Color (int, int, int)

        ( int, int, int, int )
                public java.awt.Color (int, int, int, int)

        ( int, boolean )
                public java.awt.Color (int, boolean)
```

(java.awt.color.ColorSpace, float[ ], float)
public java.awt.Color (java.awt.color.ColorSpace, float[ ], float)

Methoden:

HSBtoRGB
public static int
HSBtoRGB (float, float, float)

RGBtoHSB
public static float[ ]
RGBtoHSB (int, int, int, float[ ])

brighter
public java.awt.Color
brighter ( )

createContext
public synchronized java.awt.PaintContext
createContext (
java.awt.image.ColorModel,
java.awt.Rectangle,
java.awt.geom.Rectangle2D,
java.awt.geom.AffineTransform,
java.awt.RenderingHints)

darker
public java.awt.Color
darker ( )

decode
public static java.awt.Color
decode (java.lang.String)

equals
public boolean
equals (java.lang.Object)

getAlpha
public int
getAlpha ( )

getBlue
public int
getBlue ( )

getColor
public static java.awt.Color
getColor (java.lang.String)

getColor
public static java.awt.Color
getColor (java.lang.String, int)

getColor
public static java.awt.Color
getColor (java.lang.String, java.awt.Color)

getColorComponents
public float[ ]
getColorComponents (java.awt.color.ColorSpace, float[ ])

getColorComponents
public float[ ]
getColorComponents (float[ ])

getColorSpace
public java.awt.color.ColorSpace
getColorSpace ( )

getComponents
public float[ ]
getComponents (java.awt.color.ColorSpace, float[ ])

getComponents
public float[ ]
getComponents (float[ ])

getGreen
public int
getGreen ( )

getHSBColor
public static java.awt.Color
getHSBColor (float, float, float)

getRGB
public int
getRGB ( )

getRGBColorComponents
public float[ ]
getRGBColorComponents (float[ ])

getRGBComponents
public float[ ]
getRGBComponents (float[ ])

getRed
public int
getRed ( )

```
        getTransparency
                public int
                getTransparency (  )

        hashCode
                public int
                hashCode (  )

        toString
                public java.lang.String
                toString (  )

}
```

Viele der Methoden erkären sich häufig intuitiv über ihren (englischen) Namen. Diese Erkenntnis sollte bei der Benennung nutzereigener Klassen beachtet werden.

Anwendungen und Applets

Applikationen, die eigenständige Programme darstellen, führt der Java-Interpreter aus; dieser lädt die class-Datei, die durch die Methode main( ) als Hauptprogramm gekennzeichnet ist, welches weitere Methoden aufrufen kann.

```
public static void main(String[ ] arguments) {
  ...
   if (arguments .length > 0) {
         for (int i  = 0; i < arguments .length; i++) {
         Separierung und Verarbeitung der einzelnen Argumente ...
         }
   ...
    }
  ...
}
```

Die Haupt-Klasse muss die Deklarationen public static void aufweisen. Das String-Array in der Parameterliste der Methode main, welches beliebig benannt werden kann, übergibt mögliche (Kommandozeilen)-Argumente beim Aufruf der Applikation an diese zur weiteren Verarbeitung (Beispiel s. o.).

Java-Applets werden innerhalb eines Browsers ausgeführt, der Java unterstützt. Zu diesen Browsern gehören die aktuellen Versionen von Firefox/Mozilla, Netscape Navigator, Microsoft Internet Explorer, Opera und Sun HotJava. Applets können zusätzlich mit dem appletviewer angezeigt werden, der im Java2 Software Development Kit enthalten ist. Um ein Applet auszuführen, muss es über entsprechende HTML-Befehle in eine Webseite eingefügt werden.
z. B.

```
<object codetype="application/java"
         classid="java:___.class"
         archive="___.jar"
         width="..." height="...">
</object>
```

Das Applet wird mit der Methode init( ) initialisiert;

```
public void init( ) {
Anweisungen...
}
```

diese Methode ist vergleichbar mit der Methode main bei Java-Anwendungen; sie setzt die Deklarationen public void voraus.

Operatoren

Zuweisung

- = Der links vom Operator stehenden Variablen wird der Wert des Ausdrucks auf der rechten Seite zugewiesen.

z.B.

```
i = i+1;
```

Die Typen elementarer Variablen können über eine Zuweisung auch ineinander umgewandelt (casting) werden; dabei ist nur eine Aufwärts-umwandlung elementarer Werte in einen größeren Typ möglich:
(byte → short → integer → long , float → double)

Arithmetische Operationen

- + Addition
- - Subtraktion
- * Multiplikation
- / Division
- % Modulo (Rest einer Ganzzahldivision)

z.B.

```
int x;
x=20%7;
```

d. h. x = 6

Inkrementierung/Dekrementierung

- ++ Inkrementierung einer Variablen um 1
- -- Dekrementierung einer Variablen um 1

z.B.

```
i++
```

i=i+1

Zuweisungsoperatoren

- x += y; d. h. x = x + y
- x -= y ; d. h. x = x – y
- x *= y; d. h. x = x * y
- x /= y; d. h. x = x / y

Werden mehrere Operatoren in einem Ausdruck aufgerufen,

z.B.

```
y = 6 + 12 / 2;
```

d.h. y → 12

so geht Punkt- (Multiplikation, Division) vor Strichrechnung (Addition, Differenz).

Logische Operatoren

- && logisch AND
- || logisch OR
- ! logische Negation

z.B.

```
boolean unusual = (fleiss == 0) && (note < 2);
```

Relationale und Gleichheitsoperatoren

- > größer als
- >= größer gleich
- < kleiner als
- <= kleiner gleich
- == ist gleich
- != ungleich

z.B.

```
boolean rgo;
int x;
x = 20%7;
rgo = x==6;
```

Explizites Casting

Abwärts-Umwandlung von Datentypen oder Objekten

Objekte einer übergeordneten Klasse definieren nicht alle Verhaltensweisen, die ein Objekt einer untergeordneten Klasse besitzt. So ist es z. B. notwendig ein Graphics-Objekt in ein Graphics2D-Objekt zu konvertieren (casten), bevor bestimmte Grafikausgaben (z.B. Farbverläufe) auf dem Bildschirm möglich werden.

Bei der Wandlung eines großen Datentyps in einen kleineren Typ (z. B. float → integer) gehen mit der Konvertierung Informationen verloren, da in diesem Fall alle Nachkommastellen abgeschnitten werden. Eine Konvertierung ist aber zulässig, wenn sie explizit veranlasst, d. h. mit der folgenden Syntax erzwungen wird.

- (Typname) Wert;
- (Klassenname) Objekt;

z.B.

```
byte c;
int a,b;
c = (byte) (a*c)
```

oder

```
Graphics2D sc  = (Graphics2D) screen
```

Der Programmierer ist im Fall einer Umwandlung elementarer Variablen dafür verantwortlich, dass das Ergebnis den zulässigen Wertebereich nicht überschreitet.