Unterlagen: Praktische Informatik 2, WS 98

Prof. Dr. Manfred Schmidt-Schauß
Fachbereich Informatik
Johann Wolfgang Goethe-Universität
Postfach 11 19 32
D-60054 Frankfurt

E-mail:schauss@ki.informatik.uni-frankfurt.de
URL: http://www.ki.informatik.uni-frankfurt.de/
Tel: 069 / 798 28597 o. 28213
Fax: 069/ 798 28919

5. Mai 1998

Kapitel 1

WWW, OO und Java

1.1 Internet + WWW

Das Internet ist ein weltweiter Verbund von Geräten (Computer, Router, Gateways) (Knoten) , die mit Standleitungen oder Telefonleitungen verbunden sind und über Internetprotokolle kommunizieren.

Der Datenaustausch ist paketvermittelt: Daten werden in Pakete zerlegt und dann von Sender zum Empfänger übertragen und beim Empfänger wieder zusammengesetzt.

Intranet: lokale Netzwerke (auch globale firmeninterne), die getrennt vom Internet arbeiten.

Etwas Geschichte:

- ca. 1970: Ursprünge des Netzes sind Projekte des amerikanischen Verteidigungsministeriums.
- 1983: ARPANet, danach verschieden andere Netze (NSFNET) Starke Nutzung in Forschungseinrichtungen und Universitäten (E-Mail)
- 1988 Internet Wurm.
- 1991 /1993 Mosaic, Gopher, WWW. Die Anfänge gehen wesentlich auf Konzepte zurück, die am CERN (conseil Europèen pour la Recherche Nuclèaire) entwickelt wurden.
- 1996: 3 Millionen europäische Knoten

Die Knotenadressen (IP-Adressen) sind von der Form $n.n.n.n$, wobei n eine Zahl zwischen 1 und 255 sein kann. Z.B. ist 141.2.10.1 die Adresse des Rechners der KI. Es sind nicht alle kombinatorisch möglichen Adressen zulässig. Manche haben feste Bedeutung: 127.0.0.1. `loopback.device`, ist immer die eigene Adresse. Hier gibt es Vereinbarungen, den Adressraum zu erweitern.

Domain Name System (DNS):
Rechner (Knoten) haben eine textuelle Adresse "hera.informatik.uni-frankfurt.de", die von verschiedenen zentralen Rechnern in eine Knotenadresse umgesetzt werden kann. Als Standard wird das letzte Kürzel dem Land bzw. einem anderen Bereich zugeordnet ISO Ländercodes: Z.B. de = Deutschland, fr = Frankreich, at = Österreich , au = Australien , jp = Japan.

Andere Kürzel, die ursprünglich und weitgehend auch jetzt noch nur Knoten in den USA bezeichnen:

- com (commercial) Industrie, Firmen, usw.
- org nicht kommerzielle Organisationen (DV-bezogen)
- net (Netz bezogen)
- edu (Universitäten, Schulen, ...)
- mil (Militär, USA)
- gov (Regierung, USA)
- eu (Europa)

Die wesentlichen Aktivitäten sind:

- elektronische Post (E-mail, verschiedene Protokolle)
  Versenden und Empfangen von elektronischen Texten/Nachrichten/ Information. Einseitig, Asynchron.

  I.a. wird das Protokoll SMTP (simple mail transport protocol) verwendet. Dies funktioniert analog dem Prinzip der Briefpost: Briefe (mit Absender und Empfänger-Adresse und Daten) werden von einer Poststelle zum Verteiler gesendet, evtl. zwischengelagert, und landen im Briefkasten des Empfängers. Dieses Protokoll kann i.a. nur 7-bit ASCII-Texte behandeln, d.h. das 8te Bit wird abgeschnitten. Um damit Binärdaten zu versenden. müssen diese erst umkodiert werden (Z.B. uuencode, uudecode). Ein anderes Protokoll ist MIME (Multipurpose Internet Mail Extension), das mehr Dateitypen kennt (Typen sind: text,message, application,image, audio, video; es gibt noch Untertypen).

  Ein Problem ist unerwünschte Post (Reklame, ...). Die könnte im Extrem das ganze System zusammenbrechen lassen: Szenario: alle Firmen senden mit einem Klick ihre Reklame an viele (alle) Mailadressen. Im akademischen Bereich gibt es gewisse Kontrollen, die kommerzielle Reklamepost weitgehend ausschalten.
- News (nntp)
  Analog zu schwarzen Brettern. Alle können es lesen, wenn sie zum schwarzen Brett gehen und davor stehen. Man kann auch selbst einen Zettel festmachen. Nach einer

gewissen Zeit werden alte Nachrichten entfernt. Es gibt viele sogenannte Newsgruppen zu den verschiedensten Themen. Zu unmoderierten kann jeder seinen Senf beitragen. Moderierte Newsgruppen zensieren die Zettel (Nachrichten) vorher.

- File-Transfer (FTP = file transfer protocol)
  Zum direkten Versenden (put) und Holen (get) von Daten/Dateien/Files vom eigenen Rechner zum entfernten Rechner. Die Verbindung ist synchron. Meist wird dies genutzt unter Verwendung der Kennung `anonymous`, die nur eingeschränkte Berechtigungen hat.

- Einwählen in andere Rechner (Telnet, remote login)
  Dabei dient der eigene Rechner als Bildschirm, die Ausführung der Programme auf dem entfernten Rechner. Damit kann man z.B. seine elektronische Post auf dem Fachbereichsrechner von den USA her lesen: Zunächst mit telnet in Frankfurt sich anmelden, dann die Post lesen.

- WWW (World Wide Web, http = hyper text transport protocol)
  WWW ist ein virtuelles Netzwerk innerhalb des physikalischen Internets. Dies ist mittlerweile sehr populär geworden. Die Arbeitsweise kann man sich folgendermaßen vorstellen: Jeder Teilnehmer stellt auf seinem Rechner (Knoten) gewisse Daten (i.a. HTML (hyper text markup language)-Dokumente) zur Verfügung, die andere sich von außerhalb anschauen dürfen. Diese Daten sind i.a. als Einheiten strukturiert, die über Adressen (Links) verbunden sind und eine eigene weltweit eindeutige Adresse haben (URL). Der Benutzer (die Surferin) holt sich die Daten, wobei i.a. ein sogenannter Netz-Browser (z.B. NetScape) verwendet wird, der die Daten in Form von Seiten (graphisch gestaltet) anzeigt. Um diese graphische Gestaltung sehr flexibel zu gestalten, wurde Java (von Sun) entwickelt. Java ist ein Programmiersprache, bei deren Design die Prioritäten entsprechend gesetzt wurden: Portabilität, Sicherheit, objektorientiert, in HTML leicht verwendbar. Wir werden Java noch behandeln.

  Jede Seite (Datenpaket) hat eine Adresse und wird unter Angabe dieser Adresse geladen. Man kann das WWW als großes und offenes Hypertextsystem ansehen: Als System von vernetzten Dokumenten.

  Das Auffinden von Information kann per "Mund-zu-Mund" Propaganda geschehen, oder man sammelt solche Adressen. Suchmaschinen, die allen zugänglich sind, versuchen Adressen zu filtern, die bestimmte Stichworte enthalten.

  Hier gibt es noch viele ungeklärte (juristische, finanzielle, soziale,...) Probleme: kriminelle Aktivitäten (Kinderpornographie), politisch extreme Agitation, Copyright-Probleme, nicht einheitliche Gesetzgebung in verschiedenen Ländern, Verschlüsselung, ...)

  Wann werden Gebühren für welchen Dienst erhoben?, Reklame auf WWW-Seiten? , Besteuerung von WWW-Dienstleistungen?)

- Zeitsynchronisation (mit Laufzeit Netz und Programm): (ntp network time protocol)
  Hiermit kann ein Rechner über Netz andere Zeit-server befragen, wieviel Uhr es jetzt ist. Da man Laufzeiten (Netz, Funk, Programm) zu berücksichtigen hat, ist es nicht ganz einfach, hieraus mit genügender Genauigkeit die aktuelle Zeit zu errechnen.

- Telekonferenzen (ursprünglich schriftlich); direkte übertragung von Video, Audio Daten (Telefon)
  Dies war ursprünglich ein (fast) synchroner Austausch von Text: eine schriftliche Unterhaltung war möglich. Es gibt auch Software, die Telefonieren (Audiodaten) in Realzeit überträgt, sofern man die Übertragungskapazität hat.
- Bestellen, Bezahlen, finanzielle Transaktionen. Auch dies wird z.T. über Netzdienste abgewickelt.

1.2 Objektorientierter Programmierstil am Beispiel von Java

Im folgenden werden die Beispiele konform zur Syntax der objektorientierten Programmiersprache Java notiert, die selbst eine an C++ angelehnte Syntax hat. Allerdings erfährt Java wie jede Programmiersprache Änderungen und Erweiterungen, so daß man für genauere Informationen das entsprechende Handbuch konsultieren sollte.

Die zentrale Idee der Objektorientierten Programmiersprachen (OOP) ist die des Objekts als Strukturierung.

Objekte sind zusammengesetzte (Daten-)Einheiten, die auch als Einheit ansprechbar sind. Die Objekte kommunizieren miteinander durch Senden von Nachrichten. Veränderungen werden stets mittels sogenannten Methoden durchgeführt.

Klassen: werden in der Programmiersprache definiert. Sie definieren die Struktur von Objekten und auch der Definition der Methoden, die auf Objekte dieser Klasse anwendbar sind.

Wenn wir mit Haskell vergleichen: Klasse entspricht in etwa einem Typ bzw. einer Typklasse, ein Objekt entspricht einem konstruierten Datenobjekt. Das Senden von Nachrichten entspricht dem Funktionsaufruf. Allerdings gibt es in Java **Seiteneffekte**, (im Gegensatz zu Haskell), z.B. kann ein Objekt verändert werden, auch wenn verschiedene Verweise darauf existieren (sogenanntes. Aliasing).

1.2.1 Objekte

Ein Objekt besteht aus Daten und Methoden. Von außen ist das Innere eines Objektes nicht sichtbar, es ist gekapselt. Nur über definierte Schnittstellen ist das Innere des Objektes (seine Attribute) sichtbar. Ebenso kann das Objekt nur über solche vordefinierten Schnittstellen verändert werden.

Ein Objekt kann man sich vorstellen wie einen Satz (Verbund) in einer Datenbank, bestehend aus Attributen: ein n-Tupel, bestehend aus verschiedenen Daten.

Jedes Objekt ist Instanz einer Klasse.

1.2.2 elementare Datentypen

In Java gibt es elementare Datentypen: Zahlen (byte, short, int, long), Gleitkommazahlen (float, double), Character (char), Boolesch (boolean). Zu den elementaren Datentypen gibt es Konstanten: Zahlen im üblichen Sinne: +1, -30, Boolesche Konstanten: `true, false`. Character werden im 2-byte Uni-code dargestellt. Die Klasse String ist nicht elementar, sondern eine vordefinierte Klasse.

Es gibt OOP, in denen auch elementare Datentypen als Objekte behandelt werden.

Beispiel 1. Ein Auto hat als mögliche Attribute: Marke, Farbe, Kennzeichen, Besitzer, Baujahr, KW, Treibstoffverbrauch, Km-Stand.

Methoden:

- Erzeuge neues Objekt vom Typ Auto mit den folgenden Attributwerten: .....
- Ändere den Besitzer
- Ändere das Kennzeichen
- Berechne das Alter des Autos

1.2.3 Nachrichten

Die Anwendung von Methoden auf Objekte wird durch den Mechanismus des *Sendens von Nachrichten* durchgeführt.

Dies ist analog zur Anwendung einer Funktion auf ein Argument, wobei das erste Argument der Funktion stets das Objekt ist, dem die Nachricht gesendet wird. Das Abfragen von Attributen wird ebenfalls per Nachricht durchgeführt, wobei der Wert des Attributes als Wert der Anwendung zurückgegeben wird.

Beispiel 2. Gegeben sei ein Rechteck durch die Koordinaten seiner Eckpunkte. Das Bewegen auf dem Bildschirm könnte man durchführen mittels folgender Sequenz, die in einer Methode als Kode stehen könnte:

```
if (obj.left < 20)
    {obj.erase;                  //  Loeschen des Bildes
      obj.move(20,0);       //  Verschieben des Rechtecks (d.h. der Koordinaten)
      obj.paint                   //  Neuzeichnen
    }
```

Die Nachrichten sind jeweils "left", "erase", "move(20,0)", "paint".

1.2.4 Klassen

Klassen fassen gleichartige Objekte zusammen. Klassen verhalten sich zu Objekten wie der Begriff "Auto" zu einem konkreten Auto mit dem KZ. "B-SE-1". Das konkrete Auto ist dann eine "Instanz" (Ausprägung, Element) der Klasse.

In OOP besteht das Programmieren aus dem Definieren von Klassen, d.h. Attribute und Methoden. Die Methoden entsprechen der "normalen" Programmierung.

Klassen können hierarchisch angeordnet werden. D.h. es gibt Oberklassen und Unterklassen. Die hierarchische Anordnung soll dazu dienen, in der Oberklasse die gemeinsamen Methoden und Attribute zu definieren, die für alle Unterklassen gelten. (Statt Unterklassen sagt man auch abgeleitete Klasse).

Beispiel 3. Die Klasse `Fahrzeug` kann man analog zu `Auto` definieren. Als Unterklassen kann man `LKW, KFZ, Bus, Motorrad` definieren. Bei `LKW` kämen als Attribute hinzu: zul. Gesamtgewicht, Art der Güter, ...

Bei einem Bus: maximale Anzahl Passagiere, Anzahl Sterne, ...

Beispiel 4. Rechteck mit oberer Eck-Koordinaten, Breite und Höhe:

```
class Rechteck {
    int x,y,b,h;        // x,y Koordinaten, b,h,  Breite und Hoehe
    void paint (Graphics g) {    //Methode zum Zeichnen
       g.drawRect (x, y, b, h);  // Aufruf einer Bibliotheksfunktion drawRect
    }
}
```

1.2.5 Methoden

In einer Klasse werden sowohl die Attribute als auch die darauf möglichen Operationen definiert. Diese Operationen nennt man `Methoden`. Deren Definition ist innerhalb einer Klassendefinition.

Beispiel 5.
```
class Rechteck {
    int x,y,b,h;   //x,y Koordinaten, b,h,  Breite und Hoehe
    void paint (Graphics g) {  // Methode zum Zeichnen
       g.drawRect (x, y, b, h); //  Aufruf einer Bibliotheksfunktion drawRect
    }
    void move (int xd; int yd) {
       this.x = x + xd;
       this.y = y + yd;
    }
```

```
}
```

Die Methode `move` verändert die Eck-koordinate des Rechtecks und die Differenz.

Wenn einem Rechteck `r` die Nachricht `move` gesendet wird in der Form `r.move(20,0)`, dann wird die Anfangskoordinate des Rechtecks entsprechend der Argumente verändert.

Die Verwendung von `this` bezieht sich immer auf das Objektes, an das die Nachricht gesendet wird. In anderen OOP-Sprachen wird dies z.t. auch als `self` bezeichnet.

Sucht man die Analogie in Haskell, so würde man `r.move(20,0)` etwa als `move r 20 0` schreiben. Das Schlüsselwort `this` entspricht dann der Variablen `x` in einer Definition von `move x y z = ....`

Beachte, daß die Argumente der Nachrichten nicht nur elementare Datentypen sind, sondern auch Objekte sein können.

Typen bzw. Klassenangaben müssen in Java für alle Methoden, deren Argumente und deren Ergebnis gemacht werden. `void` steht für den Ergebnistyp einer Methode, deren Ergebnis nicht verwendet wird.

```
 void move (int xd; int yd)
```

bedeutet, daß `move` keinen Ergebnistyp hat, nur einen Seiteneffekt bewirkt, und daß die Argumenttypen `int` sein müssen.

1.2.6 Klassen und Vererbung

Klassen werden, wenn möglich und sinnvoll, hierarchisch strukturiert. D.h. man versucht diese so zu strukturieren, daß man zunächst eine möglichst allgemeine Klasse definiert, z.B. geometrischesObjekt:

```
class geometrischesObjekt {
   String Name;
   geometrischesObjekt (String Name) {this.Name = Name;}
   void paint (Graphics g);
}
```

Die Klasse Rechteck, Gerade, Kreis, Dreieck, usw kann man dann als Unterklassen definieren, die mehr Methoden und mehr Eigenschaften haben, also spezieller sind:

```
class Rechteck extends geometrischesObjekt {
    int x,y,b,h;                  //  x,y Koordinaten, b,h,  Breite und Hoehe
    void paint (Graphics g) {
       g.drawRect (x, y, b, h);
```

```
    }
    void move (int xd; int yd) {
       this.x = x + xd;
       this.y = y + yd;
    }
}
```

Eine weitere Unterklasse von Rechteck könnte ein farbiges Rechteck oder ein mit einem Muster gefülltes Rechteck sein.

```
class farbigesRechteck extends Rechteck {
    farbe int
    ....
}
```

Hierbei *erben* die Unterklassen die Attribute und Methoden der Oberklasse. Es ist möglich, neue hinzuzufügen. Die Unterklasse hat dann die Vereinigung der Methoden der Oberklasse und der in der Klasse definierten Komponenten.

Es ist auch möglich, die Methode der Oberklasse in der Unterklasse zu überschreiben. Allerdings nur für die Nutzung in dieser Unterklasse (und deren Unterklassen). Dies ist notwendig, da für spezialisiertere Klassen die Methoden weiter spezialisiert werden müssen. Z.B. Die Methode `paint` wird bei einem Rechteck nur die Linien des Rechtecks zeichnen, bei einem farbigen Rechteck die Linien und zusätzlich die Farbe.

Es ist nicht möglich, Attribute oder Methoden zu entfernen.

Zur Laufzeit kann man mit dem Operator `instanceof` feststellen, ob ein Objekt zu einer Klasse gehört: `r instanceof Rechteck`.

Oft definiert man eine Oberklasse als **abstrakte** Klasse mit dem Zugriffsrecht **abstract**. Diese Klasse darf dann keine Implementierung von Methoden enthalten, sondern nur die Deklaration des Verhaltens auf den Klassen, d.h. nur die Schnittstelle. Es ist dann auch nicht möglich, Objekte zu erzeugen, die genau zu dieser Klasse gehören. Dies geht nur für "konkrete" Unterklassen.

1.2.7 Wesentliche Eigenschaften von OOP

Das Programm besteht aus einer Menge von Klassendefinitionen, die aus Erklärungen von Attributen und Definitionen von Methoden (und Schnittstellenbeschreibungen) bestehen. Die Verwendung von Methoden geschieht wie eine Funktionsanwendung in Attributnotation. D.h. Klassen und Methoden sind das eigentliche Programm.

Spricht man von der Ausführung des Programms, dann gibt es Objekte. Diese werden innerhalb eines Methodenaufrufs zur Laufzeit (mittels new) erzeugt Das Senden von Nachrichten ist ebenfalls eine Aktion zur Laufzeit.

Vererbung Klassen können von Superklassen Methoden und Attribute erben. Dies gilt damit auch für die Objekte: auf diese kann man alle Methoden einer Superklasse anwenden.

Polymorphie Variablen können Objekte verschiedener Klassen bezeichnen. I.a. muß die Argumentvariable einer Oberklasse der aktuellen Klasse des Objekts sein, daß das Argument ist. Damit kann man Programmteile allgemein formulieren. Diese sind meist in einer Oberklasse angesiedelt. Man muß zum Zeitpunkt der Formulierung nicht wissen, welche Unterklassen zum Programm noch hinzugefügt werden.

Dynamische Bindung Gleiche Operationen (gleicher Name, gleiche Anzahl von Argumenten, gleiche Intension) die auf Objekte verschiedener Klassen angewendet werden, können verschieden programmiert sein. An der Stelle der Verwendung muss man keine Rücksicht auf die Klasse nehmen. "Dynamisch" deswegen, da man nicht immer während der Kompilierung herausfinden kann, welche Klasse das Objekt hat, d.h. welche implementierte Variante der Methode angewendet werden muß.

Das gibt es in Java nicht:

Mehrfachvererbung. D.h. eine Klasse kann nicht Unterklasse von zwei anderen unabhängigen Klassen sein. Dies vereinfacht die Kompilierung und macht die Vererbung eindeutiger. Allerdings sind Hierarchien in Anwendungsbereichen nicht immer baumförmig, so daß diese Beschränkung manchmal merkbar wird. Sie kann Kopieren von Klassen und Methoden, oder eine etwas unnatürliche Klassenhierarchie erzwingen. In Java gibt es die Möglichkeit, mittels "Interfaces", diese Beschränkung zu umgehen.

generischen Klassen Dies sind Klassen, die selbst Typ-Parameter haben können. Damit kann man z.B. Listen von Zahlen als Klasse definieren. Diese Konstruktion ist nur dann sinnvoll, wenn ein Typcheckalgorithmus in der OOP analog zum Milner- Typcheck verwendet wird. In Java kann man Listen von Objekten definieren, d.h. das sind eher heterogene Listen, wobei die Objekte in der Liste zu einer Unterklassen einer gemeinsamen Oberklasse gehören müssen.

1.2.8 Programmierhinweise

Die Erstellung eines Programms in einer OOP erfordert zunächst, die Anwendung gedanklich so zu strukturieren, daß man die Klassenstruktur festlegen kann. Diese Klassenstruktur sollte natürlich den Bereich der Anwendung möglichst gut modellieren, da man nur dann erwarten kann, daß das Programm sich robust bei Veränderungen verhält. D.h. auch bei Erweiterungen muß die Klassenstruktur nur wenig geändert werden.

Die Methoden und Attribute, die man in einem Programm benötigt, ergeben sich dann aus der benötigten Funktionalität.

Die Superklassen haben wenig Attribute und Methoden, die in der Hierarchie weiter unten stehenden Klassen haben mehr Attribute und Methoden.

1.2.9 Eigenschaften von Java

Automatische Speicherverwaltung Objekte, auf die es keine Referenz mehr gibt, werden automatisch freigegeben (garbage collection). Dies im Gegensatz zu z.B. C++.

Typsicherheit Der Kompiler überprüft die Klassenangaben auf Konsistenz.

API: Application Programmers Interface

Portabilität Es wird Wert darauf gelegt, daß die Ausführung der Programme auf allen Rechner und Rechnertypen gleichartig abläuft. Die nichtportablen Teile sind in Bibliotheken ausgelagert. Das erfordert, daß es keine Adressrechnungen gibt (die meist nicht portabel sind).

Bytecode Der Kompiler erzeugt kein lauffähiges Programm, sondern einen interpretierbaren String: den "Bytecode". Dieser wird dann von einem entsprechenden Interpreter ausgeführt. Dieser Bytecode ist portabel, d.h. auf allen Maschinen ablauffähig. Der Interpreter ist i.a. nicht portabel. Zwar entsteht durch das Interpretieren ein Geschwindigkeitsnachteil gegenüber anderen Methoden, dies ist jedoch kein Nachteil für die Anwendungen, für die Java entwickelt wurde, nämlich solche mit hohem Anteil an Netzwerk-Kommunikation und graphischer Benutzerinteraktion.

WWW Java ist konzipiert zur Programmierung von graphischen, interaktiven Programmstücken (sogenannte Applets), die von einem WWW-Browser geladen und ausgeführt werden können (der Bytecode des Applets). Man kann allerdings auch normale Programme (applications) damit schreiben, die dann von einem Interpreter ausgeführt werden können.

C++ - Ähnlichkeit Die Syntax von Java ist der von C++ sehr ähnlich, auch wenn bei der Entwicklung von Java darauf geachtet wurde, solche Eigenschaften zu vermeiden, die immer wieder Probleme verursacht haben. So gibt es keine Operationen zum expliziten Referenzieren oder Dereferenzieren von Objekten oder Daten: primitive Daten werden immer dereferenziert verwendet und Objekte und Felder immer referenziert. D.h., die sogenannte Adressrechnung ist in Java verboten

Sicherheit Das Fehlen von Referenzen und Zeigern, die vom Programm beliebig manipuliert werden können, sowie die Einschränkung der Zugriffsrechte für Applets, die über das Netzwerk kommen, tragen zur Sicherheit von Java bei.

Bibliotheken Java (JDK) hat eine Vielzahl von bereits fertigen Bibliotheken, die die Programmierung erleichtern, aber auch für einige Funktionalitäten notwendig sind. Applets kann man nur programmieren, wenn die entsprechenden Bibliotheken per import dazugeladen werden.

1.3 Programmierung in Java

1.3.1 Operatoren

$*, /, +, -$	arithmetische Infix Ops
%	modulo-Rechnung 7%3 ergibt 1
++	addiere 1 , i++ (verwende; addiere 1)
	++i (addiere 1; verwende)
$--$	subtrahiere 1 i– (verwende; subtrahiere 1)
	–i (subtrahiere 1; verwende)
$<, <=, >, >=, ==, !=$	arithmetische Infix Vergleiche
!	Negation
&, $\vert$	striktes und, oder für Boolesch
&&, $\vert\vert$	lazy und, oder
ˆ	exklusives oder
op=	Abkürzung: x op= a ist x = x op a

Es gibt noch weitere Operationen, die analog zu Registeroperationen sind: bitweises und, oder, Negation, Shift, ...

1.3.2 Anweisungen

`if`

Syntax: `if` (⟨ Ausdruck ⟩) ⟨ Anweisung ⟩ `else` ⟨ Anweisung ⟩

Hinter `if` muß ein Boolescher Ausdruck stehen. Mehrere Anweisungen, mit { und } geklammert, zählen als eine Anweisung. Das `else` kann weggelassen werden.

Beispiel 6. Beispiele für Boolesche Ausdrücke:

```
 (a == b), (a != b), (a > b).
```

Man kann auch zusammengesetzte Boolesche Ausdrücke verwenden:

```
 (a > 0  && a <= 10)
(3*a +5 < 20  ||  b == c)
```

Eine Alternative zu `if` ist die Schreibweise `a ? b : c`, die der Anweisung `if a then b else c` entspricht.

`switch`

Syntax:

`switch` (⟨ Ausdruck ⟩) { `case` ⟨ Konstante ⟩: ⟨ Anweisungen ⟩ `break` ;
...
`case` ⟨ Konstante ⟩: ⟨ Anweisungen ⟩ `break` ;
`default`: ⟨ Anweisungen ⟩
}

Analog zu C: Der Ausdruck muß eine ganze Zahl sein, die Konstanten müssen zur Kompilierzeit bekannt sein. Die break's sollten jeden Fall abschließen, können aber weggelassen werden.

`while`

Syntax:

`while` (⟨ Ausdruck ⟩) ⟨ Anweisung ⟩

Der Rumpf der Schleife wird ausgeführt, solange der Ausdruck den Wert `true` hat. Man kann mit der `continue` Anweisung ans Ende des Rumpfes springen.

`do-while`

Syntax:

`do` ⟨ Anweisung ⟩ `while` (⟨ Ausdruck ⟩)

Der Rumpf der Schleife wird ausgeführt, bis der Ausdruck den Wert `true` hat. Der Rumpf wird auf jeden Fall einmal ausgeführt.

`for`

Syntax:

`for` (⟨ Init ⟩; ⟨ Bedingung ⟩; ⟨ Ausdruck ⟩) ⟨ Anweisung ⟩

Der Rumpf der `for`-Schleife wird ausgeführt, solange der Ausdruck den Wert `true` hat. Man kann mit der `continue` Anweisung ans Ende des Rumpfes springen.

Beispiel 7. `for (int i = 0; i <= 10; i++) Anweisung;`

Diese Schleife wird für $i = 0, \ldots 10$ durchlaufen. Alternativ zum Ausdruck `i++` kann man auch $i = i + 1$ oder $i+ = 1$ schreiben.

Markierungen

Anweisungen kann man mit einer Marke (Label) versehen:

```
marke1:   while (i < 0)  i--;
```

Diese Markierung kann benutzt werden, wenn man geschachtelte Schleifen hat, und diese entweder mit `break Marke1` oder `continue Marke1` ans Ende springen will.

Konstruktoren

Die Konstruktion eines neuen Objektes mittels **new** erfordert bei der Klassendefinition eine spezielle Methode, den sogenannten *Konstruktor*, der den gleichen Namen wie die Klasse hat. Wenn Attribute sowohl in der Oberklasse als auch zusätzliche in der Unterklasse definiert sind, dann ist im Konstruktor der Zugriff auf den Konstruktor der Oberklasse mit `super` möglich.

Felder

Felder (arrays) sind Folgen von Objekten gleichen Typs. Diese können mehrdimensional definiert und benutzt werden. Wir betrachten zunächst eindimensionale Felder.

Beispiel 8.
```
int a [] ;        //     integer feld a ist erkl\"art
a =  new int [10] ;   //  integer feld a ist jetzt ansprechbar
for (int i = 0; i < a.length; i++)  a [i] = i
```

Man sieht, daß Felder vor Benutzung erst angelegt werden müssen und daß auf Felder die Methode `length` anwendbar ist, die die Länge zurückliefert.

Ein einfaches Beispiel für ein zweidimensionales Feld ist:

```
String[][] matrix = {
  {"AA","AB","AC","AD"},
  {"BA","BB","BC","BD"},
  {"CA","CB","CC","CD"}
};
```

1.3.3 Kommentare

Es gibt drei Arten von Kommentaren:

- Kommentar bis Zeilenende `//`
- Kommentare zwischen `/*` und `*/`
- Kommentare zwischen `/**` und `*/`

Kommentare der letzten Art werden vom Programm `javadoc` aus dem Programmtext extrahiert, um zur Dokumentation der Klassen zu dienen.

1.3.4 Klassenkonversion, Typkonversion, cast

Die Syntax des cast-Operators ist:

`(neuer Typ) Objekt`

Damit kann man beispielsweise schreiben:

`varfloat = (float) vardouble`

Die umgekehrte Zuweisung kann Java automatisch richtig umsetzen, da keine Information verlorengeht.

`vardouble = (float) varfloat` ist erlaubt.

Das gleiche kann man auch für Klassen verwenden: Wenn `geoObj` Oberklasse von `Rechteck` ist, dann ist die Zuweisung

`vargeoobj = varrechteck` erlaubt, der Inhalt wird nicht verändert.

Die Zuweisung

`varechteck = vargeoobj`

ist verboten, denn es könnte z.B. ein Kreisobjekt der Variablen `vargeoobj` zugeordnet sein.

Erlaubt ist: `var_rechteck = (Rechteck) var_geoobj` . Hier kann es zu einer Ausnahme zur Laufzeit kommen, wenn das Objekt ein Kreis ist.

Diese Typkonversion bei Klassen dient im wesentlichen dazu, die Methoden von Oberklassen verwendbar zu machen.

1.3.5 Ausnahmen, Exceptions

Java hat eine Behandlung von Exceptions (Ausnahmen), wie Fehler bei externen Aufrufen, Division durch 0. Die zugehörigen Schlüsselwörter sind `try, catch, throw`. Fehler, die normalerweise zum Abbruch führen, kann man Abfangen mit `catch`, nachdem man die Anweisung, die Fehler liefern könnte, mit `try` eingepackt hat. `throw` dient zum Erzeugen von programmierten Abbrüchen.

Wenn eine Methode selbst Ausnahmen erzeugen soll, dann muß dies in der Definition der Schnittstelle angezeigt werden, und zwar mit `throws Exception1, ....`

Die Verwendung von `try` und `catch`: Einem try-Block der Form `try Ausdruck` folgen catch-Blöcke: `catch exception) Anweisung` .

1.3.6 Zugriffsrechte

Die wichtigsten sind `public` und `private`. Z.B. schreibt man normalerweise:

....

```
class Rechteck extends geometrischesObjekt {
private int x, y, b, h;
public void move (int xd; int yd) {
       this.x = x + xd;
       this.y = y + yd;
    }
```

- `public` bedeutet hier, daß die Methode `move` von außerhalb benutzt werden darf. d.h. die Nachricht `move(10,10)` kann an ein Objekt der Klasse Rechteck gesendet werden.
- private bedeutet, daß die Attribute x,y,b,h nicht direkt geändert werden dürfen, und außerdem auch nicht abgefragt werden können. d.h. `r.b` ist nicht zulässig.

Weitere Zugriffsrechte gibt es im Zusammenhang mit packages, die ein Paket von Klassen sind.

1.3.7 Modifikatoren

- `static`: Ein als statisch gekennzeichnetes Attribut existiert in allen Objekten der ganzen Klasse nur einmal. Für Attribute wirkt dies wie die Erklärung zu einer globalen Variable für diese Klasse. Hier kann man z.B. die Anzahl aller Objekte sich merken, oder eine Liste aller Objekte mitführen. Allerdings ist das in Konkurrenz zum Garbage Collector, der nichtreferenzierte Objekte löschen darf. Durch die pure Anwesenheit in dieser Liste werden die Objekte dann referenziert.

  Statische Methoden kann man als "Klassenmethoden" ansehen, die man benutzen kann, ohne daß ein Objekt vorhanden ist, d.h. die Variable `this` kann darin nicht verwendet werden. Die zugehörigen Methoden können benutzt werden, ohne daß ein Objekt existiert, indem man diese aufruft mit: Klassennamen. Methodenname.
- `final`: Bewirkt, angewandt auf eine Datenkomponente, daß deren Wert als nicht überschreibbar angesehen wird. Damit kann Java einerseits feste Konstanten definieren, andererseits Oberklassen und Methoden, die nicht durch Unterklassen verändert werden dürfen.

1.3.8 Applets und Applikationen

In Java kann man zwei verschiedene Arten von Anwendungen schreiben:

applets Dazu muss im Programmcode die import-Anweisung `import java.applet.*;` stehen. Der Name der Datei und der definierten Klasse, die die ausgeführte Klasse des applets ist, sollten übereinstimmen. Diese ausgezeichnete Klasse muss eine Unterklasse von `Applet` sein. Hierdurch sind automatisch zwei Methoden definiert, die für die Klasse überschrieben werden sollten: `init(), paint (Graphics g)`. Das Applet kann dann in eine HTML-Datei eingebunden werden mit dem Schlüsselwort `applet`. Bei Aufruf der Hauptklasse werden diese aufgerufen und erzeugen dann eine entsprechende Zeichnung bzw. interaktive graphische Anwendung. Wie oben schon erwähnt, hat ein applet normalerweise Beschränkungen beim Zugriff auf Files der Festplatte des Klienten.

Zur Laufzeit ist das Applet jeweils in einen Kontext eingebunden, der plattformabhängig ist.

Applikationen Diese haben in der Hauptklasse eine Methode `main`:

`public static void main (String args [])` { ... die bei Aufruf der Klasse dann ausgeführt wird. Deren Argument ist standardmäßig ein Feld von Strings: die Parameter, die bei Aufruf übergeben werden.

Beispiel 9. `// ebene geometrische Objekte`

```
//   erster File:  Punkt.java

public class Punkt {
   int x,y;
   Punkt (int x, int y) {
     this.x = x; this.y = y;
   }
}

//   zweiter File:  geoobjtest.java

import java.applet.*;
import java.awt.*;


public class geoobjtest extends Applet {
    GeoObjekt [] geos;
    Punkt p1,p2,p3,p4,m1;
    GeoObjekt lin;
    public void init () {
      p1 = new  Punkt (10,10);
      p2 = new  Punkt (50,50);
      p3 = new  Punkt (100,10);
      p4 = new  Punkt (80,120);
      lin = new Linie (p1,p4);
      m1 = new Punkt (50,70);
      geos = new GeoObjekt[3];
      geos[0] = new Kreis (30, m1);
```

```
      geos[1] = lin;
      geos[2] = new Dreieck (p1,p2,p3);
    }
    public void paint (Graphics g) {
       for (int i = 0;i < geos.length;i++)
          geos[i].zeichne(g);
    }
}
abstract class GeoObjekt {
  public abstract void zeichne (Graphics g);
  public abstract boolean innen (Punkt p);
}

class Rechteck extends GeoObjekt {
    public Punkt lu;
    public int b,h;    // x,y Koord , b,h,  Breite,Hoehe
    Rechteck (Punkt p,int b, int h) {
      this.lu = p; this.b = b; this.h = h;
    }
    public void zeichne (Graphics g) { // Methode zum Zeichnen
       g.drawRect (lu.x, lu.y, b, h);      // Aufruf Bibfu drawRect
    }
    public boolean innen (Punkt p) {
         return this.lu.x <= p.x && p.x <= this.lu.x+b &&
                    this.lu.y <= p.y && p.y <= this.lu.y+h;
    }
}

class Kreis extends GeoObjekt {
    public int r;    // Radius
    public Punkt m;   //  Mittelpunkts
    Kreis (int r, Punkt m) {
      this.m = m; this.r = r;
    }
    public void zeichne (Graphics g) {
       g.drawArc (m.x-r, m.y-r, 2*r, 2*r, 0, 360);
                // Aufruf  der Bib fu drawArc
    }
    public boolean innen (Punkt p) {
         return (m.x-p.x)*(m.x-p.x) +
                (m.y-p.y)*(m.y-p.y) <= r*r;
    }
}
```

```
class Linie extends GeoObjekt {
    public Punkt a,b;
    Linie (Punkt a, Punkt b) {
       this.a = a; this.b = b;
    }
    public void zeichne (Graphics g) {
       g.drawLine (a.x, a.y, b.x, b.y);
    }
    public boolean innen (Punkt p) {
      Dreieck dr;
      dr =  new  Dreieck (p,this.a, this.b);
      return (p == this.a || p == this.b ||
              dr.entartet() && (b.x-p.x)*(p.x-a.x) > 0
              && (b.y-p.y)*(p.y-a.y) > 0);
    }
}


class Dreieck extends GeoObjekt {
    public Punkt p1,p2,p3;                    // Koordinaten der Ecken
    Dreieck (Punkt p1, Punkt p2, Punkt p3) {
       this.p1 = p1; this.p2 = p2; this.p3 = p3;
    }
    public void zeichne (Graphics g) {
       g.drawLine (p1.x, p1.y, p2.x, p2.y);
       g.drawLine (p2.x, p2.y, p3.x, p3.y);
       g.drawLine (p3.x, p3.y, p1.x, p1.y);
    }
    public boolean entartet () {
      return (p3.x-p1.x)*(p2.y-p1.y)
        == (p2.x-p1.x)*(p3.y-p1.y);
    }

    public boolean innen (Punkt r) {
        return true;
    }

}
```

1.3.9 Klassenbibliotheken in Java

java.lang Basispaket, das z.B. Zahlen implementiert.

java.util Einige extra Basisfunktionalitäten: Hashtabellen, Datum, Random, Vector, enumeration (Liste)

java. applet Erzeugt sogenannte Applets. Eine Anwendung, die ein Applet werden soll, muß als Unterklasse von applet definiert sein.

java.awt Abstract window toolkit:

java.io Dient zur Verarbeitung (Lesen, Schreiben) von Files, Puffern, Pipes, Druckern, usw.

java.net Verarbeitung von Netzzugriffen, Verbindungen, Lesen, Laden von Ressourcen (URLs).

1.3.10 Mathematische Bibliothek

`public final class Math`
Die Klasse kann keine Unterklassen haben, und kein Methode kann überschrieben werden. Alle Methoden und Variablen sind static. Die Ausnahmebehandlung wie Überlauf, Division durch 0, usw. werden auf verschiedenen Plattformen möglicherweise verschieden gehandhabt (d.h. nicht portabel).

Es gibt folgende Konstante / Funktionen / Methoden:

E,PI	
ceil(x), floor(x), rint(x, round(x)	ganzzahliger Anteil, bzw. Rundungen
cos(x), sin(x), tan()	trigonometrische Funktionen
acos(x), asin(x), atan(x)	Umkehrfunktionen
exp(x), log(x), sqrt(x) , pow(x,y)	Einige mathematische Funktionen
max(x,y), min(x,y), abs(x)	mathematische Funktionen
random()	Ergibt eine Zufallszahl zwischen 0.0 und 1.0

1.4 Graphische Programmierung (GUIs)

Da Java eine (relativ) einfache Möglichkeit bereitstellt, graphische Benutzeroberflächen zu programmieren, können wir kurz in die Programmierung und Handhabung von graphischen Benutzeroberflächen einführen.

Da die Benutzung eines Computers heute die Benutzung einer graphischen Benutzeroberfläche fast zwangsläufig zur Folge hat, ist jeder schon mal Benutzer gewesen. Diese Oberflächen zeichnen sich durch die Verwendung von Fenstern aus, die verschiedenen Anwendungen zugeordnet sind, die aktiv oder inaktiv sein können, und die visuell im Hintergrund oder im Vordergrund sein können. Ein weiteres Kennzeichen ist die parallele Verwendung der Tastatur und einer Maus mit 1,2 oder 3 Tasten.

Es gibt eine Vielzahl von graphischen Elementen, die verwendet werden können: Fenster mit Schiebeleisten, waagrechte/senkrechte Menüs, Radioknöpfe, Symbole zum Anklicken (Icons), usw.

Übrigens, die Verwendung von GUIs ist noch nicht so alt. Die Verbreitung startete vor ca. 15 Jahren. Zu diesem Zeitpunkt war die Verwendung von zeichenbasierten Bildschirmen und Interaktion mit Kommandoeingabe in eine Kommandozeile vorherrschend. Auch jetzt gibt es diesen Modus der Interaktion noch als Basis, z.B. bei PC als DOS Kommandozeile, oder als Kommandozeile bei Unix. Bei Großrechnern (Mainframes) ist dies noch der interne Standard, obwohl auch hier graphische Benutzeroberflächen in vielen Fällen zur Interaktion mit dem Rechner verwendet werden.

Die große Akzeptanz beim Benutzer kommt durch die einfache Bedienbarkeit, bei der man sich keine Kommandosequenzen und Syntax merken muß. Zudem kommt die graphische Oberfläche der menschlichen Wahrnehmung entgegen, die eher an Bildern orientiert ist.

Möglich wurden die graphischen Oberflächen erst durch schnelle Rechner und grössere Speicher, die eine schnelle Verarbeitung und Darstellung ermöglichten. Vergleicht man die Vernetzung bei Großrechner: > 1000 Bildschirme an einem Zentralrechner (meist über eigene dezentrale Rechner verbunden), und wie sie aktuell über Netze (Internet) verbunden sind, so erkennt man, daß die graphische Benutzeroberfläche im Zusammenhang mit Vernetzung erzwingt, daß das Terminal selbst lokal über die notwendige (Rechen+Speicher)Kapazität verfügen muß.

1.4.1 Programmierung, Ereignissteuerung

Im Gegensatz zur Verwendung einer Kommandozeile, bei der ein entsprechendes Programm genau den Inhalt (einen String) der Kommandozeile sieht und entsprechend verarbeitet, muß eine programmierte graphisches Benutzeroberfläche und entsprechend die Anwendungsprogramme mit einer Mehrzahl von Interaktionen und auch Eingabemöglichkeiten umgehen können.

Z.B. muß ein Textverarbeitungsprogramm unter GUI i.a. die Markierung eines Textstückes ermöglichen. Dazu notwendig sind: Das Programm muss wissen, was im Moment sichtbar ist, es muss herausfinden, wo die Maustaste gedrückt wurde, wohin sie gezogen wurde und wo die Maustaste wieder losgelassen wurde.

Die interne Methode der Mitteilung an das Programm geschieht über einen Strom von Ereignissen, die vom System, Betriebssystem oder dem GUI-System erzeugt werden und an das Benutzerprogramm gesendet werden. Dieser Strom ist nach der zeitlichen Reihenfolge des Auftretens der Ereignisse geordnet.

Das Prinzip der Portabilität hier einzuhalten ist nicht ganz unproblematisch, da die verschiedenen GUIs und Systeme bezüglich der erzeugten Ereignisse nicht ganz identisch sind.

In Java wird die Portabilität gewährleistet (bzw. unterstützt) durch sogenannte "peers"-Klassen. Diese stellen eine Zwischenschicht zwischen dem GUI-System der Plattform und dem Java-GUI dar. Sie gewährleisten, daß das Programmierer sich (fast) nicht um die tatsächliche Plattform kümmern muß. Ein Beispiel für einen Unterschied ist die verschiedene Anzahl der Maustasten, die 1, 2, oder 3 sein kann, je nach System.

1.4.2 GUIs in Java

Betrachten wir jetzt mal die Möglichkeiten in Java. Normalerweise ist die graphische Anwendung ein Applet, das durch eine Klasse definiert wird, die zur Unterklasse von applet erklärt wird.

Das Bild, das der Benutzer sieht, stellen wir uns vor als einen flachen, graphischen und farbigen Bildschirm, der aus ca. 1000 X 1000 Bildpunkten, sogenannten Pixeln besteht. Dies ergibt in natürlicher Weise ein Cartesisches Koordinatensystem, das jeden Punkt in der Form (x,y) ansprechbar macht. Normalerweise gibt es einen Cursor, das ist eine aktuelle Position der Tastatur und einen Mausposition, die sich innerhalb der Fläche befinden.

Es gibt eine reichliche Anzahl vordefinierter Fenstertypen und graphischer Elemente:

Oberste Klasse ist `Component`. Unterklassen sind:

Button Ein Knopf zum Anklicken.

Canvas Ein Zeichenfläche.

Checkbox Ein Text und ein Kästchen zum Ankreuzen.

Choice Ein Auswahlmenü mit Kurztexten.

Container abstrakte Klasse, mit der hierarchisch ein Fenster in Unterfenster gegliedert werden kann. Die Unterklassen sind:

- **Panel** Fenster ohne irgendwelche Rahmenobjekte.
- **Window** mit Unterklassen *Frame, Dialog*

Label Textanzeigefeld

List Senkrechte anklickbare Menüleiste.

Scrollbar Schiebeleiste

Textcomponent Textanzeige

Applet ist Unterklasse von Panel, was nur bedeutet, daß deren Methoden einem applet sofort zur verfügung stehen.

Zur Klasse *Component* gibt es eine Menge von Methoden, die verschieden Aspekte handhaben: Einige Beispiele seien hier aufgeführt.

Layout Die aktuelle Größe und Position kann ermittelt und verändert werden. `bounds()`, `inside()`, `location()`, `move()`, `size()`, ...

Farbe Des Hintergrunds, Vordergrunds `getBackground()`, `setForeground()`, ...

Ereignisse Ermitteln und Handhabung der durch den Benutzer verursachten Ereignisse `action()`, `gotFocus()`, `keyDown()`, `mouseDown()`, `mouseEnter()`, `mouseUp()`, `handleEvent()`, `postEvent()`, `ldots`.

Zeichnen innerhalb der Komponente, Neuzeichnen des Bildes. `paint()`, `repaint()`, `update()`

Status Feststellen, ob Komponente aktiv, usw.

Bei Definition einer Unterklasse können bzw. müssen bestimmte Methoden in diesen Klasse überschrieben werden.

Bemerkungen

Die Klasse `Container` hat die Methode `add`, mit der man neue Komponenten hinzufügen kann. Diese werden normalerweise vom zugehörigen Layout-manager an bestimmte, vordefinierte Positionen plaziert: Möglich sind u.a. `North, West, South, East, Center, null`.

Die Ereignisse, die vom Benutzer und System generiert werden, werden in der Objekthierarchie von unten nach oben durchgereicht, bis es eine Methode `handle_event` gibt, die diese einfängt. Diese Methode kann das Ereignis auch abweisen (bzw. weiterreichen, indem `false` zurückgegeben wird.

Programmierhinweise: Generell gilt: Man muß das an der Oberfläche sichtbare als Modell möglichst vollständig in Datenstrukturen ablegen, damit das sichtbare Bild rekonstruierbar ist und damit der Benutzer und das Programm die gleiche Sicht der Dinge haben.

Ist die zu programmierende Anwendung komplex, so ist die Handhabung und das Verarbeiten aller möglichen Ereignisfolgen eine ebenfalls komplexe und schwierige Aufgabe, da man mit allem rechnen muß und auf jedes Ereignis eine sinnvolle Aktion gestartet werden soll.

Um eine komplexere (gute, professionelle) Anwendung (Applet) in Java zu schreiben, ist ein aktuelles Handbuch notwendig. Denn man muß eine Unzahl von Klassen, die Hierarchie, deren Methoden, die richtige Verwendung und auch die zu behandelnde Ausnahmen kennen. Eine Einarbeitung und Orientierung wird meist erleichtert durch vorgegebene Beispiele im JDK (oder auch aus anderen Quellen).

Eine Anwendung sollte anwenderfreundlich sein und einfach und zuverlässig wirken und auch sein. Das erfordert oft etwas Nachdenken und gutes Strukturieren. Zum Beispiel ist im nachfolgenden Zeichenprogramm extra Aufwand getrieben worden, um während des Zeichnens die aktuelle Strecke(Ellipse) sichtbar zu machen. Analog verhält es sich mit Fehlermeldungen. Um diese richtig zuzuordnen und zu formulieren, sollte man sich in die Rolle eines Benutzer versetzen, der keine Ahnung von der internen Verarbeitung hat, und der nicht abgeschreckt werden soll.

Beispiel 10. Dieses Beispiel ist ein Applet, das eine Zeichenfläche realisiert, auf der man Strecken und Ellipsen zeichnen kann. Es gibt einen Schalter, der mit `choice` realisiert ist, mit dem man von Strecke auf Ellipse umschalten kann.

Während des Zeichenvorgangs wird das jeweils aktuelle Objekt gezeichnet, bis es durch Loslassen der Maustaste bestätigt wird.

```
//   veraenderte Version eines JDK Beispiels

import java.awt.*;
import java.applet.*;

import java.util.Vector;

public class DrawTest extends Applet {
    public void init() {
setLayout(new BorderLayout());
DrawPanel dp = new DrawPanel();
add("Center", dp);
add("South", new SKWahl(dp));
dp.init();
    }

    public boolean handleEvent(Event e) {
switch (e.id) {
  case Event.WINDOW_DESTROY:
    System.exit(0);
    return true;
  default:
    return false;
}
    }

    public static void main(String args[]) {
Frame f = new Frame("DrawTest");
DrawTest drawTest = new DrawTest();
drawTest.init();
drawTest.start();

f.add("Center", drawTest);
```

```
f.resize(300, 300);
f.show();
    }
}

class DrawPanel extends Panel {
    Vector lines = new Vector();
    Vector streckeOderKreis = new Vector();
    int x1,y1;
    boolean ziehen;
    boolean strecke;
    int x2,y2;

    public void init() {
       ziehen = false;
       strecke = true;
    }
    public DrawPanel() {
setBackground(Color.white);
    }

   public void setStrecke() {
       strecke = true;
   }
   public void setKreis() {
       strecke = false;
   }
   public boolean handleEvent(Event e) {
switch (e.id) {
  case Event.MOUSE_DOWN:
x1 = x2 = e.x;
y1 = y2 = e.y;
ziehen = true;
    return true;
  case Event.MOUSE_UP:
lines.addElement(new Rectangle(x1, y1, e.x, e.y));
      // leichter Missbrauch von rectangle
streckeOderKreis.addElement(new Boolean(strecke));
ziehen = false;
    repaint();
    return true;
  case Event.MOUSE_DRAG:
x2 = e.x;
```

```
y2 = e.y;
    repaint();
    return true;
  case Event.WINDOW_DESTROY:
    System.exit(0);
    return true;
  default:
    return false;
}
    }


    public void paint(Graphics g) {
int np = lines.size();

/* Zeichne alle Strecken und Ellipsen */
for (int i=0; i < np; i++) {
    Rectangle p = (Rectangle)lines.elementAt(i);
         if (((Boolean) streckeOderKreis.elementAt(i))
                        .booleanValue())
      g.drawLine(p.x, p.y, p.width, p.height);
   else
   g.drawOval(p.x, p.y, p.width-p.x, p.height-p.y);
}
if (ziehen) {
  if (strecke)
      {g.drawLine(x1, y1, x2, y2);}
  else {
        g.drawOval(x1,y1,x2-x1,y2-y1);
  }
    }
    }
}


class SKWahl extends Panel {
    DrawPanel target;

    public SKWahl(DrawPanel target) {
this.target = target;
setLayout(new FlowLayout());
Choice SoderK = new Choice();
SoderK.addItem("Strecke");
```

```
SoderK.addItem("Ellipse");
SoderK.setBackground(Color.lightGray);
add(SoderK);
    }

    public void paint(Graphics g) {
Rectangle r = bounds();
g.setColor(Color.lightGray);
g.draw3DRect(0, 0, r.width, r.height, false);
    }

    public boolean action(Event e, Object arg) {
if (e.target instanceof Choice) {
    String choice = (String)arg;
    if (choice.equals("Strecke")) {
   target.setStrecke();
    } else if (choice.equals("Ellipse")) {
   target.setKreis();
    }
}
return true;
    }
}
```