Eine Einführung in die Grundlagen der Objektorientierten Programmierung

Dr. Jürgen Czischke, Dr. Georg Dick, Horst Hildebrecht, Ludger Humbert, Werner Ueding, Klaus Wallos.

Bearbeitung: Dieter Lindenberg

Version 2017.9 **JAVA**

Inhalt

Installationen

Installationsanleitung für JSDK

Zum Programmieren in Java benötigt man neben einem Texteditor zum Eintippen des Programms noch einen Übersetzer (engl. Compiler), der den Text in eine ausführbare Form bringt. Ein solcher Compiler ist Bestandteil des *Java Development Kits* (JDK), manchmal auch **Java-Software-Development-Kit (JSDK)** genannt. Eventuell ist dies auf deinem Rechner schon vorhanden.

Die Installation wird hier für Windows beschrieben. Für Mac-OS-X ist das nicht nötig, weil die Java-Entwicklungsumgebung schon im Betriebssystem integriert ist.

1. Auf http://www.oracle.com/technetwork/java/javase/downloads/index.html die neueste Version des JDK herunterladen.

2. Mit Administratorrechten jdk-8u144-windows-x64.exe (64-Bit-Version, Versionsnummer 144 im Jahr 2017) oder jdk-8u144-windows-i586.exe (32-Bit-Version, Versionsnummer 144 im Jahr 2017) zur Installation starten.

3. Custom Setup: Demos und Source Code kann man weglassen.
Install to: C:\Program Files\Java\jdk1.8.0_144\ Dieser Name (im Juli 2017) kann bei neueren Versionen leicht geändert sein.
Zielverzeichnis eventuell anpassen. Auf alle Fälle den Verzeichnisnamen merken! – Next. Die Registrierung ist nicht unbedingt erforderlich.

4. In Windows 7/8/10 (als Administrator anmelden):
Start - Systemsteuerung -System - Einstellungen ändern - Erweitert – Umgebungsvariablen
Die System-Variable *Path* bearbeiten, d.h. den Pfad C:\Program Files\Java\jdk1.8.0_144\bin (bzw. den aktuellen Namen) nach einem Semikolon ergänzen. Das bewirkt, dass Java-Dateien automatisch auch in diesem Java-Ordner gesucht werden. Beispiel:
Der Wert der Variablen Path sei SystemRoot%\system32;%SystemRoot%
Dann ergänzt man den Wert zu
%SystemRoot%\system32;%SystemRoot%;C:\Program Files\Java\jdk1.8.0_144\bin
Achtung: Das Semikolon vor C: nicht vergessen! Und den aktuellen Namen einsetzen!

5. Test: Start - Programme - (Zubehör) – Eingabeaufforderung
Die Eingabe von JAVA.EXE sollte das Programm starten können (es erscheint eventuell eine Optionenliste). Wenn keine Fehlermeldung kommt, dass das Programm nicht gefunden wurde, ist die *Java-Runtime-Umgebung* richtig installiert.

Die Eingabe von JAVA**C**.EXE sollte den Java-Compiler starten (es erscheint eine Optionenliste). Wenn keine Fehlermeldung kommt, dass das Programm nicht gefunden wurde, ist auch das *Development-Kit* richtig installiert.

Installation von BlueJ (für Windows)

1. Man findet *BlueJ* auf der Seite http://www.bluej.org . Die aktuelle Version (September 2017) trägt die Versionsnummer 4.1.1. Hat man, wie oben dargestellt, das JDK installiert, muss nur noch der „BlueJ Installer“ heruntergeladen werden.
Zur Installation ist die Datei *bluej-411.msi* (oder aktuellerer Name) zu starten. Es empfiehlt sich, das Verzeichnis *C:\Program Files\BlueJ* als Installationsverzeichnis zu wählen, um diverse Anpassungen (siehe unten) zu minimieren.

2. Unter *Tools-Preferences-Interface* kann die Sprache der Oberfläche auf Deutsch umgestellt werden.

Installation der Java-Dokumentation
Auf http://www.oracle.com/technetwork/java/javase/downloads/index.html findet man weiter unten den Download-Link von *Java SE8 Documentation*. Die heruntergeladene Datei mit dem (September 2017) aktuellen Namen *jdk-8u144-docs-all.zip* enthält in gezippter Form das Verzeichnis *docs*. Dieses (extrahierte) Verzeichnis muss in den BlueJ-Ordner gelegt werden.

Installation der SuM-Bibliothek (Stifte und Mäuse)

1. Auf der Seite http://www.nili-software.de/sum/ findet man die Datei *SuMWin.zip* .
Nach dem Herunterladen und Entpacken dieser Datei entstehen im Verzeichnis *SuMWin7.5* die Verzeichnisse *Bibs*, *doc*, *german* und die Datei *bluej.defs* .

2. Den Ordner *doc* lege man in das BlueJ-Verzeinis (dort, wo BlueJ installiert wurde).

3. Der Ordner *german* ersetzt den gleichnamigen Ordner im Unterverzeichnis *lib* des BlueJ-Ordners.

4. Man kopiert alle Dateien in *Bibs\in userlib* in das BlueJ-Verzeichnis *lib\userlib*.

5. Man kopiert die Datei *bluej.defs* in das BlueJ-Verzeichnis *lib* und ersetzt die dortige gleichnamige Datei.
Wenn das BlueJ-Verzeichnis *C:\Program Files\BlueJ* ist, dann ist man fertig. Wurde BlueJ in ein anderes Verzeichnis installiert, dann muss man in der Datei *bluej.defs* alle Bezüge auf *C:\Program Files\BlueJ* entsprechend anpassen.

Die Klasse *Bildschirm*

Ein Bildschirm ist das Modell des angeschlossenen Computerbildschirms. Es entspricht einem Fenster auf dem Computerbildschirm. Auf ihm kann mit Stiften gezeichnet werden. Zu diesem Zweck ist die Zeichenebene auf dem Bildschirm mit einem Koordinatensystem versehen, dessen Ursprung sich in der oberen linken Ecke der Zeichenebene befindet und dessen Achsen horizontal nach rechts und vertikal nach unten gerichtet sind. Die Einheit ist ein *Pixel (picture element)*.
Hinweis: Wenn der Bildschirm von einem anderen Fenster überdeckt wird, so wird der überdeckte Bildschirminhalt gelöscht und leider nicht wieder automatisch aktualisiert.

Bildschirm
- Breite - Höhe - Ort - Hintergrundfarbe
+ ! init() + ? breite() + ! gibFrei() + ? hoehe() + ? holeGanzeZahl() + ? holeText() + ? holeZahl() + ! loescheAlles() + ! nachVorn() + ! setzeFarbe()

Jede Klasse besitzt (mindestens) einen Konstruktor, mit dessen Hilfe ein Objekt dieser Klasse erzeugt wird. Ein Konstruktor ist ein besonderer öffentlicher (public) Dienst, der den gleichen Namen besitzt wie die Klasse. Daran wird dieser Dienst auch als Konstruktor erkannt. Beachte: im Quelltext fehlt beim Konstruktor das Wort *void*.
Der Konstruktor ist praktisch der Initialisierungsteil eines Klassenobjektes. Deshalb wird er im Klassendiagramm auch oft als *init()* bezeichnet.

Konstruktoren:

Bildschirm()
Der Bildschirm ist mit seiner Zeichenebene mit maximaler Größe initialisiert.

Bildschirm(int pBreite, int pHoehe)
Der Bildschirm ist mit seiner Zeichenebene initialisiert. Der obere, linke Eckpunkt besitzt die Computer-Monitor-Koordinaten (0; 0). Die beiden Parameter bestimmen Breite und Höhe.

Bildschirm(int pLinks, int pOben, int pBreite, int pHoehe)
Der Bildschirm ist mit seiner Zeichenebene initialisiert. Die Parameter pLinks und pOben sind Koordinaten des Computer-Monitors; sie legen den oberen, linken Eckpunkt des Bildschirmfensters fest. Die beiden anderen Koordinaten bestimmen die Breite und Höhe des Bildschirmfensters.

Methoden:

int **breite**()
liefert die Breite der Zeichenebene

void **gibFrei**()
Nachdem eine Taste oder der Mausknopf gedrückt wurde, steht der Bildschirm nicht mehr zur Verfügung, d.h. die Zeichenebene verschwindet.

int **hoehe**()
liefert die Höhe der Zeichenebene.

int **holeGanzeZahl**()
liefert eine ganze Zahl, die mit einem Eingabedialog gelesen wird. Falls die Eingabe keine ganze Zahl ist, wird 0 zurückgegeben.

int **holeGanzeZahl**(String pMeldung)
liefert eine ganze Zahl, die mit einem Eingabedialog gelesen wird. Falls die Eingabe keine ganze Zahl ist, wird 0 zurückgegeben. pMeldung wird als

Eingabeaufforderung angezeigt.

String **holeText**()
liefert einen Text, der mit einem Eingabedialog gelesen wird.

String **holeText**(String pMeldung)
liefert einen Text, der mit einem Eingabedialog gelesen wird. pMeldung wird als Eingabeaufforderung angezeigt.

double **holeZahl**()
liefert eine (Komma-) Zahl, die mit einem Eingabedialog gelesen wird. Falls die Eingabe keine Zahl ist, wird 0.0 zurückgegeben.

double **holeZahl**(String pMeldung)
liefert eine (Komma-) Zahl, die mit einem Eingabedialog gelesen wird. Falls die Eingabe keine Zahl ist, wird 0.0 zurückgegeben. pMeldung wird als Eingabeaufforderung angezeigt.

void **loescheAlles**()
Die Zeichenebene ist danach leer.

void **nachVorn**()
macht das Bildschirmfenster zum vordersten Fenster

void **setzeFarbe**(int pFarbe)
ändert die Hintergrundfarbe der Zeichenebene. Alte Zeichnungen auf dem Bildschirm werden gelöscht.

Bemerkung:
Es gibt eine eigene Klasse namens ***Farbe***, welche im Prinzip nur 13 sog. Klassenkonstanten vom Typ *int* zur Verfügung stellt. Die Namen dieser Konstanten entsprechen den deutschen Farbnamen. Deren Werte sind die Zahlen 0 bis 12. Klassenkonstanten werden üblicherweise nur mit Großbuchstaben geschrieben. Die Zuordnung steht in der folgenden Tabelle.
Um auf eine Klassenkonstante zugreifen zu können, benötigt man kein Objekt der Klasse. Man braucht dazu nur den Klassennamen und den Namen der Konstanten.

Beispiel: Die beiden folgenden Anweisungen sind in ihrer Wirkung identisch:

```
derBildschirm.setzeFarbe(1);
derBildschirm.setzeFarbe(Farbe.BLAU);
```

0	SCHWARZ
1	BLAU
2	CYAN
3	DUNKELGRAU
4	GRAU
5	GRUEN
6	HELLGRAU
7	MAGENTA
8	ORANGE
9	PINK
10	ROT
11	WEISS
12	GELB

Aufgaben

1. Erzeuge einen Bildschirm der Breite 400 und der Höhe 200, welcher einigermaßen mittig auf dem Monitor dargestellt wird!

2. Erzeuge einen Bildschirm mit blauer Hintergrundfarbe!

3. Der Benutzer wird um die Eingabe einer ganzen Zahl gebeten. Daraufhin wird die Hintergrundfarbe des Bildschirms entsprechend dargestellt.

4. Ermittle die zu den Zahlen 13 bis 24 gehörenden Farben und stelle eine entsprechende Tabelle auf!

Lösungen

Aufgabe 1-2-3

Starte *BlueJ* und wähle im Menü *Projekt* die Option *neues Projekt*! Nenne dieses neue Projekt *pAufgaben*! Als Speicherort wähle deinen persönlichen, dafür vorgesehenen Ordner!
Ein größeres Projekt enthält üblicherweise mehrere Klassen. Jede von uns in nächster Zeit programmierte Aufgabe entspricht einer eigenen Klasse. Es macht also Sinn, all unsere nächsten Aufgaben in einem einzigen Projekt namens *pAufgaben* zu sammeln.

Erzeuge eine neue Klasse mit dem Namen *KAufgabe123*! Der Klassenname muss sich vom Projektnamen unterscheiden, ansonsten kann es später zu Fehlermeldungen kommen. Klassennamen beginnen üblicherweise mit einem Großbuchstaben. Als Klassenart wähle *Klasse*! Diese Wahl legt nur fest, welches Code-Grundgerüst zuerst für die Klasse verwendet wird. Es kann problemlos verändert werden.

Anschließend sieht man im Mittelteil des Hauptfensters von BlueJ ein Rechteck (beschriftet mit *KAufgabe123*). Dieses Icon stellt die Klasse dar, aus der unser Programm (für die Aufgaben 1-3) besteht bzw. bestehen soll. Bei größeren Aufgabenstellungen kommen üblicherweise weitere Icons für entsprechende Hilfsklassen hinzu.

Um den Quelltext einer Klasse zu bearbeiten, klicke doppelt auf deren Icon. Alternative: Rechtsklick auf das Klassen-Icon und Auswahl der Option *Bearbeiten*.
Man kann nun die Implementierung einer Klasse sehen.

In dem Quelltext sind bereits mehrere Kommentare vorgegeben. Kommentare sind Bemerkungen bzw. Texte, die dem Leser des Programms das Verständnis erleichtern sollen. Sie können natürlich nicht in die Maschinensprache des Rechners übersetzt werden.

Einzeilige Bemerkungen beginnen mit // . Sie enden am Ende der Zeile.
Mehrzeilige Bemerkungen stehen zwischen den Zeichen /* und */ .
BlueJ fertigt auch selbstständig Klassendokumentationen an. Alles, was hierin aufgenommen werden soll, muss zwischen den Zeichen /** und */ stehen.

Für sehr kleine Programme wie z.B. die Übungsaufgaben für absolute Programmieranfänger lohnt sich das Einfügen von Bemerkungen noch nicht wirklich. Bei größeren Programmen sind Bemerkungen allerdings sehr wichtig.

```
import sum.kern.*; // vorgegebene Klassen aus SuM werden benötigt
/**
 * @author Dieter Lindenberg
 * @version 2017
 */
public class KAufgabe123
{
   // Objekte
   Bildschirm derBildschirm;

   // Konstruktor
   public KAufgabe123()
   {
     derBildschirm = new Bildschirm(100,200);
     derBildschirm.nachVorn();
   }

   // Dienste
   public void fuehreAus()
   {
     int n;
     n = derBildschirm.holeGanzeZahl("Gib Zahl
                                zwischen 0 und 20 ein!");
     derBildschirm.setzeFarbe(n);


     // Aufräumen
     derBildschirm.gibFrei();
   }
} // Ende der Klasse
```

Nach dem Übersetzen dieses (hoffentlich fehlerfreien) Quelltextes schließt man den Editor und man sieht im *BlueJ*-Hauptfenster das Icon dieser Klasse ohne Querstreifen. Wenn man den Editor schließt, wird der Quelltext automatisch gespeichert.

Man kann ein Popupmenü mit Befehlen erhalten, die auf eine Klasse anwendbar sind, indem man mit der rechten Maustaste auf das Klassenicon klickt. Die Befehle, die im Menü oben angezeigt werden, sind die Befehle, die durch diese Klasse bereitgestellt werden. Weiter unten folgen die Operationen, die durchdie *BlueJ*-Umgebung bereitgestellt werden.

Wir wollen zunächst ein Objekt der Klasse *KAufgabe123* erstellen. Klicke dazu mit der rechten Maustaste auf das *KAufgabe123*-Icon. Wähle jetzt den Konstruktor der Klasse *KAufgabe123*.

Im nun erscheinenden Dialogfeld kann man einen Namen für das zu erstellende Objekt eingeben. Gleichzeitig wird ein Name dafür vorgeschlagen. Dieser ist momentan ausreichend, also klicke jetzt *OK*. Ein Objekt der Klasse *KAufgabe123* wird erstellt. Alles, was im Quelltext im Konstruktor steht, wird jetzt ausgeführt. Also in unserem Fall wird nun bloß ein kleiner Bildschirm am oberen, linken Rand des Monitors sichtbar. Dies ist ein Objekt unserer Klasse *KAufgabe123*.
Gleichzeitig wird auf der Objektleiste von *BlueJ* ein Icon dieses gerade erzeugten Objektes abgelegt.

Nun kann man die öffentlichen Methoden unseres Objektes ausführen. Klicke mit der rechten Maustaste auf das Objekt-Icon und ein Popupmenü mit den Objektoperationen öffnet sich. Das Menü zeigt die Methoden, die für dieses Objekt verfügbar sind und weitere spezielle Operationen, die durch die *BlueJ*-Umgebung bereitgestellt werden.

Nach Anklicken der Methode ***fuehreAus()*** wird man nach einer Zahl gefragt und die zu dieser Zahl gehörige Hintergrundfarbe des Objektfensters wird eingestellt.

Das Programm wird beendet, indem man das entsprechende Objektfenster schließt (z.B. durch Tastendruck oder Mausklick, vgl. die Bildschirmmethode *gibFrei()*!).

Die Klasse *Stift*

Stift
Position # Richtung # Zustand # Modus
+ ! init() + ! bewegeBis() + ! bewegeUm() + ! dreheBis() + ! dreheUm() + ! dreheZu() + ! gibFrei() + ! hoch() + ? hPosition() + ? istUnten() + ! normal() + ! radiere() + ! runter() + ! schreibeZahl() + ! schreibeText() + ? vPosition() + ! wechsle() + ? winkel() + ! zeichneKreis() + ! zeichneRechteck()

Der Stift ist ein Werkzeug, das sich auf dem Bildschirm bewegen kann. Er befindet sich stets auf einer genau definierten Position des Bildschirms, die durch Zeichenkoordinaten (horizontal nach rechts, vertikal nach unten) angegeben wird, und zeigt in eine Richtung, die durch Winkel beschrieben wird (0° entspricht der Richtung nach rechts, Drehsinn ist mathematisch positiv).

Der Stift kennt zwei Zustände: Ist der Stift abgesenkt (runter) und bewegt er sich über den Bildschirm, so hinterlässt er eine Spur, die von einem Zeichenmodus abhängig ist. Ist der Stift angehoben (hoch), hinterlässt er keine Spur.

Beim Zeichnen kennt der Stift drei Modi:

- Normal: der Stift zeichnet eine Linie in der Stiftfarbe;
- Wechseln: der Stift zeichnet eine Linie, wobei die Untergrundfarbe in die Stiftfarbe und die Stiftfarbe in die Untergrundfarbe geändert wird;
- Radieren: der Stift zeichnet eine Linie in der Farbe des Untergrunds.

Konstruktor:

Stift()
Der Stift wird initialisiert. Die Zeichenebene steht zur Verfügung und der Stift befindet sich angehoben oben links an Position (0,0) mit Richtung 0° im normalen Zeichenmodus.

Methoden:

void **bewegeBis**(double pH, double pV)
Der Stift wird unabhängig von seiner vorherigen Position auf die durch die Parameter angegebene Position bewegt. Die Winkelstellung des Stiftes hat sich nicht geändert.

void **bewegeUm**(double pDistanz)
Der Stift wird von seiner aktuellen Position in die aktuelle Richtung bewegt. Der Parameter gibt die Länge der zurückgelegten Strecke an.

void **dreheBis**(double pWinkel)
Der Stift wird unabhängig von seiner vorherigen Richtung auf die durch pWinkel angegebene Winkelgröße gedreht.

void **dreheUm**(double pWinkel)
Der Stift wird ausgehend von seiner jetzigen Richtung um die durch pWinkel angegebene Winkelgröße im mathematisch positiven Sinne weitergedreht.

void **dreheZu**(double pWohinH, double pWohinV)
Der Stift wird unabhängig von seiner vorherigen Richtung in die Richtung des Punktes gedreht, dessen Koordinaten übergeben werden.

void **gibFrei**()
Der Stift wird freigegeben.

void **hoch**()
Der Stift wird angehoben.

double **hPosition**()
liefert die horizontale Koordinate der aktuellen Stiftposition.

boolean **istUnten**()
liefert *true*, wenn der Stift abgesenkt ist, ansonsten *false*.

void **normal**()
Der Stift arbeitet danach im Normalmodus.

void **radiere**()
Der Stift arbeitet danach im Radiermodus.

void **runter**()
Der Stift wird abgesenkt.

Bemerkung: bei allen folgenden Schreibmethoden wird oberhalb der aktuellen Stiftposition geschrieben. Falls der Stift also z.B. die Position (0, 0) hat, wird man nichts sehen!
Außerdem schreibt der Stift auf den Bildschirm, egal wie dieser aussieht, d.h. der Untergrund wird nicht erst gelöscht. Wenn man zweimal auf dieselbe Stelle schreibt, ist beides nicht lesbar.

void **schreibeText**(char pZeichen)
Der Stift schreibt das angegebenen Zeichen auf die Zeichenebene unter Verwendung seines aktuellen Zeichenmodus unabhängig vom Zustand. Die aktuelle Stiftposition war die linke, untere Ecke des Zeichens. Die neue Stiftposition ist die rechte, untere Ecke des Zeichens.

void **schreibeText**(String pText)
Der Stift schreibt den angegebenen Text auf die Zeichenebene unter Verwendung seines aktuellen Zeichenmodus unabhängig vom Zustand. Die aktuelle Stiftposition war die linke, untere Ecke des Textes. Die neue Stiftposition ist die rechte, untere Ecke des Textes.

void **schreibeZahl**(double pZahl)
Der Stift schreibt die angegebene Zahl auf die Zeichenebene unter Verwendung seines aktuellen Zeichenmodus unabhängig vom Zustand. Die aktuelle Stiftposition war die linke, untere Ecke der Zahl. Die neue Stiftposition ist die rechte, untere Ecke der Zahl.

double **vPosition**()
liefert die vertikale Koordinate der aktuellen Stiftposition.

void **wechsle**()
Der Stift arbeitet danach im Wechselmodus.

double **winkel**()
liefert die aktuelle Bewegungsrichtung (in Grad) des Stifts.

void **zeichneKreis**(double pRadius)
Der Stift zeichnet unabhängig von seinem Zustand im aktuellen Zeichenmodus einen Kreis mit der aktuellen Position als Mittelpunkt und dem angegebenen Radius. Die Position und die Richtung des Stiftes sind unverändert.

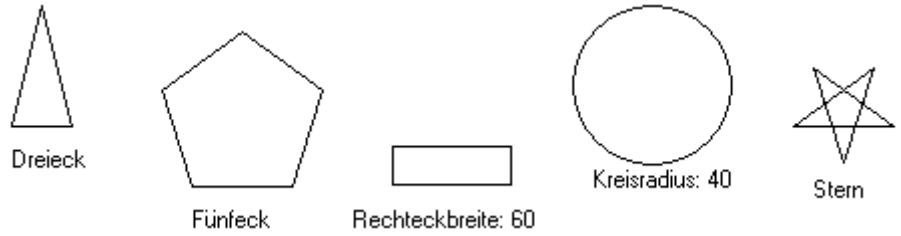
void **zeichneRechteck**(double pBreite, double pHoehe)
Der Stift zeichnet unabhängig von seinem Zustand im aktuellen Zeichenmodus ein achsenparalleles Rechteck mit der aktuellen Position als linker, oberer Ecke und der angegebenen Breite und Höhe. Die Position und die Richtung des Stiftes sind unverändert.

Aufgaben

1. Nach der Eingabeaufforderung „Gib deinen Namen ein!“ für die Variable *name* erscheint auf dem Bildschirm der Begrüßungstext (zum Beispiel) „Hallo Annabell!“

Hinweise: Beachte, wo genau der Stift schreibt (siehe Dokumentation)!
Man kann Strings hintereinanderhängen: Beispiel: `"Hallo" + name + "!"`

2. Zeichne folgende Figuren (mit zugehörigem Text) auf den Bildschirm!



Lösung Aufgabe 17_2:

```
import sum.kern.*;
/**
 * @author Dieter Lindenberg
 * @version 2017
 */
public class KAufgabe17_2
{
    // Objekte
    Bildschirm derBildschirm;
    Stift meinStift;

    // Konstruktor
    public KAufgabe18_2()
    {
        derBildschirm = new Bildschirm(500,200);
        derBildschirm.nachVorn();
        meinStift = new Stift();
    }

    // Dienste
    public void fuehreAus()
    {
       // Aktionsteil
       meinStift.bewegeBis(10,90);
       meinStift.schreibeText("Dreieck");
       meinStift.bewegeBis(10,70);
       meinStift.runter();
       meinStift.bewegeBis(40,70);
       meinStift.bewegeBis(25,10);
       meinStift.bewegeBis(10,70);
       meinStift.hoch();

       meinStift.bewegeBis(100,120);
       meinStift.schreibeText("Fünfeck");
       meinStift.bewegeBis(100,100);
       meinStift.runter();
       meinStift.bewegeUm(50);
```

```
meinStift.dreheUm(72);
meinStift.bewegeUm(50);
meinStift.dreheUm(72);
meinStift.bewegeUm(50);
meinStift.dreheUm(72);
meinStift.bewegeUm(50);
meinStift.dreheUm(72);
meinStift.bewegeUm(50);
meinStift.dreheUm(72);
meinStift.hoch();

meinStift.bewegeBis(180,120);
meinStift.schreibeText("Rechteckbreite: ");
meinStift.schreibeZahl(60);
meinStift.bewegeBis(200,80);
meinStift.zeichneRechteck(60,20);

meinStift.bewegeBis(300,110);
meinStift.schreibeText("Kreisradius: ");
meinStift.schreibeZahl(40);
meinStift.bewegeBis(330,50);

meinStift.zeichneKreis(40);

meinStift.bewegeBis(410,105);
meinStift.schreibeText("Stern");
meinStift.bewegeBis(400,70);
meinStift.runter();
meinStift.bewegeUm(50);
meinStift.dreheUm(144);
meinStift.bewegeUm(50);
meinStift.dreheUm(144);
meinStift.bewegeUm(50);
meinStift.dreheUm(144);
meinStift.bewegeUm(50);
meinStift.dreheUm(144);
meinStift.bewegeUm(50);
meinStift.dreheUm(144);
```

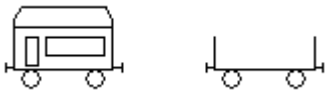
```
        meinStift.hoch();

        // Aufraeumen
        meinStift.gibFrei();
        derBildschirm.gibFrei();
    }
}
```

Aufgaben

1. Zeichne ein symmetrisches Trapez, einen Drachen und eine Raute!

2. Zeichne einen einzigen Punkt (als Kreis mit Radius 1 Pixel) an der Stelle (500, 50)!

3. Zeichne ein Haus, ein Auto und die beiden folgenden Waggons!



- Ein Waggon wird am linken Puffer ganz links in der Mitte beginnend gezeichnet.
- Die Räder haben einen Radius von 5 Punkten (pixel).
- Der Wagen hat eine Gesamtlänge (einschließlich Puffer) von ungefähr 60 Pixel.

4. Die Breite und die Höhe deines Bildschirmes sollen von deinem Programm ermittelt werden und dein Stift soll diese Daten auf dem Bildschirm beginnend an der Stelle (10; 20) ausgeben.

5. Bringe den Stift an eine beliebige Stelle auf dem Bildschirm. Der Stift soll seine Orts- und Winkelkoordinaten auf dem Schirm ausgeben, z.B. in folgender Form: (512.0 | 120.0 | 25°).
Nach der Ausgabe (bei der sich der Stift ja bewegt) soll er zu eben diesem gerade von ihm angegeben Ort zurückkehren (und natürlich auch dieselbe Winkeleinstellung wie vorher haben).

6. Bringe den Stift in die Mitte des Bildschirms unabhängig davon, welche Ausmaße der Bildschirm hat. Danach soll ein Strich gezeichnet werden. Der Benutzer wird gefragt, wohin.

a) es wird nach den Koordinaten des Zielpunktes gefragt (zwei Fragen: Zuerst nach der horizontalen, dann nach der vertikalen Koordinate).

b) es wird nach der Richtung (Winkel) und der Länge des Striches gefragt.

Lösungen

Aufgabe 3 Güterwaggon

```
public void fuehreAus()
 {
   meinStift.hoch();
   meinStift.bewegeBis(20,200); // Mitte linker Puffer

   meinStift.dreheBis(90); // linker Puffer wird gezeichnet
   meinStift.bewegeUm(10);
   meinStift.dreheBis(-90);
   meinStift.runter();
   meinStift.bewegeUm(20);
   meinStift.dreheBis(90);
   meinStift.hoch();
   meinStift.bewegeUm(10);
   meinStift.dreheBis(0);
   meinStift.runter();
   meinStift.bewegeUm(10);

   meinStift.dreheBis(-90);  // Kasten
   meinStift.hoch();
   meinStift.bewegeUm(10);
   meinStift.dreheBis(90);   // linke Kante
   meinStift.runter();
   meinStift.bewegeUm(50);
   meinStift.dreheBis(-90);
   meinStift.hoch();
   meinStift.bewegeUm(50);
   meinStift.dreheBis(0);  // untere Kante
   meinStift.runter();
   meinStift.bewegeUm(200);
   meinStift.dreheBis(90);  // rechte Kante
   meinStift.bewegeUm(50);
   meinStift.dreheBis(-90);
   meinStift.hoch();
   meinStift.bewegeUm(40);
```

```
  meinStift.dreheBis(0);   // rechter Puffer
  meinStift.runter();
  meinStift.bewegeUm(10);
  meinStift.dreheBis(90);
  meinStift.hoch();
  meinStift.bewegeUm(10);
  meinStift.dreheBis(-90);
  meinStift.runter();
  meinStift.bewegeUm(20);

  meinStift.hoch();   // linkes Rad
  meinStift.dreheBis(180);
  meinStift.bewegeUm(180);
  meinStift.hoch();
  meinStift.dreheBis(270);
  meinStift.bewegeUm(15);
  meinStift.zeichneKreis(15);

  meinStift.hoch();   // rechtes Rad
  meinStift.dreheBis(0);
  meinStift.bewegeUm(140);
  meinStift.zeichneKreis(15);


 // Aufräumen
 meinStift.gibFrei();
 derBildschirm.gibFrei();
} // Ende von fuehreAus()
```

Aufgabe 4

```
public void fuehreAus()
 {
  double b = derBildschirm.breite();
  double h = derBildschirm.hoehe();
  meinStift.bewegeBis(10,20);

  meinStift.schreibeText("Breite: "+b+"   Höhe: "+h);

  // Aufräumen
  meinStift.gibFrei();
  derBildschirm.gibFrei();
 }
```

Aufgabe 5

```
public void fuehreAus()
 {
  meinStift.bewegeBis(111,59);
  double h = meinStift.hPosition();
  double v = meinStift.vPosition();
  double w = meinStift.winkel();
  meinStift.schreibeText("("+h+" | "+v+" | "+w+"°)");
  meinStift.bewegeBis(h, v);

  // Aufräumen
  meinStift.gibFrei();
  derBildschirm.gibFrei();
 }
```

Aufgabe 6a

```
public void fuehreAus()
 {
  meinStift.bewegeBis(derBildschirm.breite()/2,
                              derBildschirm.hoehe()/2);
  double x = derBildschirm.holeZahl("x-Koordinate des
                                        Zielpunktes?");
  double y = derBildschirm.holeZahl("y-Koordinate des
                                        Zielpunktes?");
  meinStift.runter();
  meinStift.bewegeBis(x, y);
  // Aufräumen
  meinStift.gibFrei();
  derBildschirm.gibFrei();
 }
```

Aufgabe 6b

```
public void fuehreAus()
 {
  meinStift.bewegeBis(derBildschirm.breite()/2,
                              derBildschirm.hoehe()/2);
  double w = derBildschirm.holeZahl("Winkel?");
  double l = derBildschirm.holeZahl("Länge?");
  meinStift.runter();
  meinStift.dreheBis(w);
  meinStift.bewegeUm(l);
  // Aufräumen
  meinStift.gibFrei();
  derBildschirm.gibFrei();
 }
```

Prozeduren

Prozeduren sind Unterprogramme. Die vier Begriffe Prozedur, Funktion, Methode und Dienst werden oft gleichbedeutend verwandt. Man fasst üblicherweise längeren Programmtext zu einer Prozedur zusammen, insbesondere dann, wenn dieser Programmtext mehr als einmal eingesetzt wird. Prozeduren, welche ein Ergebnis liefern, werden auch Funktionen genannt. Alle Methoden (= Dienste) einer Klasse werden durch eine (oder mehrere) Prozeduren (oder Funktionen) realisiert. Aber nicht jede kleine Hilfsprozedur wird gleich Methode bzw. Dienst genannt.

Als Beispiel werden wir eine Prozedur implementieren (= programmieren), welche ein gleichseitiges Dreieck zeichnet.

```
import sum.kern.*;

public class aufgabe
{
    // Objekte
    Bildschirm derBildschirm;
    Stift meinStift;

    // Konstruktor
    public aufgabe()
    {
        derBildschirm = new Bildschirm(500,300);
        meinStift = new Stift();
    }

    // Dienste
    /**
     * Es wird ein gleichseitiges Dreieck gezeichnet. Nach dem Zeichnen
     * befindet sich der Stift wieder in exakt derselben Position und Richtung
     * wie vorher.
     */
```

```
private void zeichneDreieck()
{
    meinStift.runter();
    meinStift.bewegeUm(40);
    meinStift.dreheUm(120);
    meinStift.bewegeUm(40);
    meinStift.dreheUm(120);
    meinStift.bewegeUm(40);
    meinStift.dreheUm(120);
    meinStift.hoch();
}


public void fuehreAus()
{
    // Aktionsteil
    meinStift.bewegeBis(100,85);
    zeichneDreieck();
    meinStift.bewegeBis(300,200);
    meinStift.dreheUm(-30);
    zeichneDreieck();

    // Aufräumen
    meinStift.gibFrei();
    derBildschirm.gibFrei();
}
```

Aufgaben

1. Warum ist es nicht sinnvoll, in obiger Dreiecksprozedur den Befehl *meinStift.bewegeBis(…)* zu benutzen?

Hinweis:
Bei jeder der folgenden Prozeduren sollte sich die Prozedur die Koordinaten des Anfangspunktes und den Anfangswinkel merken. Das ist mit den Methoden der Klasse *Stift* leicht möglich. Am Ende der Prozedur sollte man grundsätzlich zum Anfangspunkt und zum Anfangswinkel zurückkehren.

2. Implementiere eine Prozedur namens *zeichneHaus*! Zeichne anschließend mehrere Häuser auf dem Bildschirm!

3. Schreibe eine Prozedur, welche das Haus des Nikolaus zeichnet! Das Zeichnen wird einfacher, wenn man als Dach ein gleichseitiges Dreieck verwendet. Achte darauf, dass jede Strecke nur einmal gezeichnet wird!

4. Schreibe eine Prozedur, welche ein rechtwinkliges, gleichschenkliges Dreieck ABC zeichnet! Der rechte Winkel soll bei B liegen. Es muss möglich sein, dass die Grundseite eine beliebige Richtung besitzt (also nicht unbedingt horizontal verläuft).

5. Schreibe eine Prozedur, welche ein rechtwinkliges, nicht gleichschenkliges Dreieck ABC zeichnet! Der rechte Winkel soll bei B liegen. Es muss möglich sein, dass die Grundseite eine beliebige Richtung besitzt (also nicht unbedingt horizontal verläuft).

Lösungen

Aufgabe 28_3 (Haus des Nikolaus)

```
import sum.kern.*;
……..
public class Nikolausaufgabe
{
……..
 // Dienste
 private void baueHaus()
 {
  double xUntenLinks = meinStift.hPosition();
  double yUntenLinks = meinStift.vPosition();
  double winkel = meinStift.winkel();
  meinStift.runter();
  meinStift.dreheBis(90);
  meinStift.bewegeUm(80);
  meinStift.dreheBis(0);
  meinStift.bewegeUm(50);
  meinStift.dreheBis(120);
  meinStift.bewegeUm(50);
  meinStift.dreheBis(-120);
  meinStift.bewegeUm(50);
  meinStift.bewegeBis(xUntenLinks + 50, yUntenLinks);
  meinStift.bewegeBis(xUntenLinks, yUntenLinks);
  meinStift.bewegeBis(xUntenLinks+50,yUntenLinks-80);
  meinStift.bewegeBis(xUntenLinks + 50, yUntenLinks);

  meinStift.hoch();
  meinStift.bewegeBis(xUntenLinks, yUntenLinks);
  meinStift.dreheBis(winkel);
 }

public void fuehreAus()
 {
  // Aktionsteil
  meinStift.bewegeBis(100, 300);
  baueHaus();
```

```
  meinStift.bewegeBis(200, 200);
  baueHaus();
  // Aufräumen
  meinStift.gibFrei();
  derBildschirm.gibFrei();
 }
}
```

Aufgabe 28_4

```
 private void zeichneDreieck()
  {
   double x = meinStift.hPosition();
   double y = meinStift.vPosition();
   double w = meinStift.winkel();
   meinStift.runter();
   meinStift.bewegeUm(40);
   meinStift.dreheUm(90);
   meinStift.bewegeUm(40);
   meinStift.bewegeBis(x,y);
   meinStift.dreheUm(-90);
   meinStift.hoch();
  }

 public void fuehreAus()
  {
   meinStift.bewegeBis(100,85);
   zeichneDreieck();

   meinStift.bewegeBis(300,200);
   meinStift.dreheUm(-30);
   zeichneDreieck();

   // Aufräumen
   meinStift.gibFrei();
   derBildschirm.gibFrei();
 }
```

Prozeduren mit Parametern

Bevor man die oben beschriebene Prozedur *zeichneDreieck()* aufrufen konnte, musste man bekanntlich den Stift erst zu dem zukünftigen Ort des Dreiecks bewegen. Diese Ortsfestlegung kann man jedoch auch gleichzeitig mit dem Prozeduraufruf angeben. Das geschieht mithilfe von Parametern. Im Folgenden sieht man die Änderungen gegenüber dem Programm auf Seite 30:

```
……….
    private void zeichneDreieck(double px, double py)
    {
        meinStift.bewegeBis(px, py);
        meinStift.runter();
        meinStift.bewegeUm(40);
        meinStift.dreheUm(120);
        meinStift.bewegeUm(40);
        meinStift.dreheUm(120);
        meinStift.bewegeUm(40);
        meinStift.dreheUm(120);
        meinStift.hoch();
    }


    public void fuehreAus()
    {
        // Aktionsteil
        zeichneDreieck(100,85);
        meinStift.dreheUm(-30);
        zeichneDreieck(300, 200);

        // Aufräumen
……..................
    }
```

Man könnte natürlich auch noch mehr Parameter übergeben, z.B. die Länge der Grundseite und die Höhe des Dreiecks. Das folgende Programm zeichnet zwei gleichschenklige Dreiecke, deren Grundseite horizontal verläuft.

```
.....
private void zeichneDreieck(double px, double py,
                                    double pl, double ph)
    {
        meinStift.bewegeBis(px, py);
        meinStift.runter();
        meinStift.bewegeBis(px+pl, py);
        meinStift.bewegeBis(px+pl/2, py-ph);
        meinStift.bewegeBis(px, py);
        meinStift.hoch();
    }

    public void fuehreAus()
    {
        // Aktionsteil
        zeichneDreieck(100,85, 50, 20);
        meinStift.dreheUm(-30); // unwirksam
        zeichneDreieck(300, 200, 50, 20);

        // Aufräumen
.....
    }
```

Aufgaben

1. Implementiere die Methode
drawCircle(double pxm, double pym, double pr), welche als Parameter die Koordinaten des Kreismittelpunktes und den Radius besitzt!

2. Implementiere die Methode
drawRectangle(double px, double py, double pb, double ph), welche als Parameter die Koordinaten des linken, unteren Eckpunktes, sowie die Breite und Höhe besitzt!

3. Implementiere die Methode
drawHouse(double px, double py, double pb, double ph), welche als Parameter den linken unteren Eckpunkt, die Breite und die Gesamthöhe besitzt!

4. Implementiere die Methode *zeichneDreieck(double px, double py, double prichtung, double pa, double pc, double pbetta)*, welche als Parameter die Koordinaten des Eckpunktes A, die Richtung der Grundseite c, die Längen der Seiten a und c und den eingeschlossenen Winkel β enthält).

5. Implementiere die Methode
baueNikolausHaus(double px, double py, double pb, double ph), welche das Haus des Nikolaus zeichnet! Die Parameter geben den Punkt unten links und die Breite und Höhe an. Das Zeichnen wird einfacher, wenn man als Dach ein gleichseitiges Dreieck verwendet. Achte darauf, dass jede Strecke nur einmal gezeichnet wird!

Lösungen

Aufgabe 33_3

```
.....
// Dienste
private void drawHouse(double px, double py, double
pb, double ph)
{
 // Sicherung
 double stiftPositionX = meinStift.hPosition();
 double stiftPositionY = meinStift.vPosition();

 meinStift.hoch();
 meinStift.bewegeBis(px, py - 0.8*ph);
 meinStift.zeichneRechteck(pb, 0.8*ph);
 meinStift.runter();
 meinStift.bewegeBis(px + pb/2, py - ph);   // Dach
 meinStift.bewegeBis(px + pb, py - 0.8*ph);

 meinStift.hoch();   // Tür
 meinStift.bewegeBis(px + 0.3*pb, py - 0.3*ph);
 meinStift.zeichneRechteck(0.2*pb, 0.3*ph);

 // Fenster
 meinStift.bewegeBis(px + 0.6*pb, py - 0.7*ph);
 meinStift.zeichneRechteck(0.2*pb, 0.2*ph);

 meinStift.hoch();
 meinStift.bewegeBis(stiftPositionX, stiftPositionY);
}

public void fuehreAus()
 {
  .....
  drawHouse(100,200,60,100);
  drawHouse(300,300,40,80);
  .....
}
```

Die Klasse *Maus*

Maus
- Position - Zustand
+ ! init() + ? doppelKlick() + ! gibFrei() + ? hPosition() + ? istGedrueckt() + ? vPosition()

Eine Maus realisiert die Mauseingabe des verwendeten Computers. Diese ist gekennzeichnet durch die aktuelle Position der Maus auf dem Bildschirm des Computers und die Betätigung der Maustaste zu einem bestimmten Zeitpunkt.

Konstruktor:

Maus()
Die Maus wird initialisiert.

Methoden:

boolean **doppelKlick**()
Prüft, ob ein Doppelklick stattgefunden hat.

void **gibFrei**()
Die Maus steht nicht mehr zur Verfügung

int **hPosition**()
liefert die gegenwärtige horizontale Koordinate der Position der Maus auf dem Bildschirm, unabhängig davon, ob die Maus gedrückt wurde.

boolean **istGedrueckt**()
Prüft, ob eine Maustaste im Moment gedrückt ist.

int **vPosition**()
liefert die gegenwärtige vertikale Koordinate der Position der Maus auf dem Bildschirm, unabhängig davon, ob die Maus gedrückt wurde.

Bedingungen

```
if (meineMaus.istGedrueckt()) meinStift.hoch();

int note = 4;
if (note == 6) meinStift.schreibeText("ungenügend");

String s = derBildschirm.holeText("wie heißt du?");
if (s.equals("Klaus")) meinStift.schreibeText("Hallo");
```

Sollen, falls die Bedingung erfüllt ist, mehrere Befehle ausgeführt werden, so müssen diese in *geschweiften Klammern* eingeschlossen werden:

```
if (meineMaus.hPosition() > 400)
  {
   meinStift.bewegeBis(meineMaus.hPosition(),
                               meineMaus.vPosition());
   meinStift.zeichneKreis(30);
  }
```

if-Abfragen können geschachtelt werden:

```
if (meineMaus.doppelKlick())
  {
   meinStift.bewegeBis(meineMaus.hPosition(),
                               meineMaus.vPosition());
   if (meineMaus.vPosition() > 200)
     {
      meinStift.zeichneKreis(30);
      meinStift.zeichneRechteck(40, 40);
     }
  }
```

Man kann auch eine sog. **zweiseitige Bedingung** angeben:

```
if (meineMaus.vPosition() >= 500)
  {
   meinStift.hoch();
   meinStift.bewegeBis(meineMaus.hPosition(),
                               meineMaus.vPosition());
   meinStift.zeichneKreis(10);
  }
else meinStift.zeichneRechteck(20,50);
```

```
if (meineMaus.vPosition() >= 50) meinStift.hoch();
else
  {
   meinStift.runter();
   meinStift.bewegeBis(70,80);
   meinStift.hoch();
  }
```

Bei geschachtelten Abfragen bezieht sich die else-Anweisung immer auf die letzte **freie** if-Abfrage. Vergleiche die beiden folgenden Anweisungsblöcke! Was bewirken sie jeweils bei folgenden Stiftkoordinaten:
a) (100; 100); b) (100; 500) c) (500; 100) d) (500; 500)

Beispiel 1:

```
if (meinStift.hPosition() >= 400)
   if (meinStift.vPosition() > 400)
      meinStift.zeichneKreis(20);
   else meinStift.zeichneRechteck(30,30);
meinStift.bewegeBis(0,0);
```

Beispiel 2:

```
if (meinStift.hPosition() >= 400)
   {
    if (meinStift.vPosition() > 400)
       meinStift.zeichneKreis(20);
   }
else meinStift.zeichneRechteck(30,30);
meinStift.bewegeBis(0,0);
```

Interessant: An diesen beiden Beispielen sieht man, dass die geschweiften Klammern nicht nur Befehle zusammenfassen, sondern diese Befehle auch als einen eigenen, nicht mehr „freien“ Teil (sog. Block) kennzeichnen. Auch lokale Variablen, die innerhalb eines solchen Blockes deklariert werden, gelten nur innerhalb dieses Blockes.

Aufgaben

39_1 Nach der Eingabeaufforderung „Wie viel ist 5 + 7 ?“ für die Variable *ergebnis* erscheint auf dem Bildschirm entweder das Wort „richtig“ oder das Wort „falsch“.

39_2 Nach der Eingabeaufforderung „Gib das Passwort ein!“ für die Variable *code* erscheint auf dem Bildschirm entweder das Wort „OK“ oder der Text „falsches Passwort“. Das richtige Passwort soll „Informatik“ sein. Die Kontrolle erfolgt mit **`if (code.equals(„Informatik“))`**…

39_3 Der Benutzer wird aufgefordert, eine Zahl einzugeben. Falls die Zahl 1 eingegeben wurde, wird der Bildschirmhintergrund blau gefärbt.

39_4 Der Benutzer wird aufgefordert, eine Zahl einzugeben. Falls die Zahl 1 eingegeben wurde, wird der Bildschirmhintergrund blau gefärbt, andernfalls rot.

39_5 Der Benutzer wird aufgefordert, eine beliebige, ganze Zahl einzugeben. Bei Eingabe von 1 wird der Bildschirmhintergrund rot gefärbt, bei Eingabe von 2 gelb, ansonsten blau.
Farbcodes:

```
0 = schwarz, 1 = blau, 2 = türkis, 3 = grau,
4 = hellgrau, 5 = hellgrün, 6 = hellgrau,
7 = violett, 8 = gelb, 9 = rosa, 10 = rot
```

Lösungen

Aufgabe 39_1

```
public void fuehreAus()
 {
   double ergebnis;
   meinStift.bewegeBis(10, 20);
   ergebnis =
      derBildschirm.holeZahl("Wie viel ist 5 + 7 ?");
   if (ergebnis == 12)
      meinStift.schreibeText("richtig");
   else meinStift.schreibeText("falsch");
   // Aufräumen
   meinStift.gibFrei();
   derBildschirm.gibFrei();
 }
```

Aufgabe 39_2

```
public void fuehreAus()
 {
  String code;
  meinStift.bewegeBis(10, 20);
  code = derBildschirm.holeText("Gib das Passwort ein!");
  if (code.equals("Informatik"))
    meinStift.schreibeText("OK");
  else meinStift.schreibeText("falsches Passwort");
  // Aufräumen
  meinStift.gibFrei();
  derBildschirm.gibFrei();
 }
```

Aufgabe 39_5

```
public void fuehreAus()
 {
  int n;
  n = derBildschirm.holeGanzeZahl("ganze Zahl
                                        eingeben!");
  if (n == 1) derBildschirm.setzeFarbe(10);
  else {
        if(n == 2) derBildschirm.setzeFarbe(8);
        else derBildschirm.setzeFarbe(1);
       }
  // Aufräumen
  meinStift.gibFrei();
  derBildschirm.gibFrei();
}
```

Logische Operatoren *Nicht, Und, Oder*

```
if (!meineMaus.istGedrueckt())
  meinStift.bewegeBis(meineMaus.hPosition(),
                                  meineMaus.vPosition());
```

```
if (!(meineMaus.hPosition() > 300 ))
  meinStift.bewegeBis(meineMaus.hPosition(),
                                  meineMaus.vPosition());
```

Beachte: hinter NOT muss ein Wahrheitswert stehen.
meineMaus.hPosition() ist alleine kein Wahrheitswert. Erst die Auswertung des obigen Klammerausdrucks liefert einen Wahrheitswert.

```
if ((meineMaus.hPosition() == 300 ) &&
                                meineMaus.istGedrueckt())
  meinStift.bewegeBis(20,30);
```

Der Stift wird im obigen Beispiel nur bewegt, wenn beide Bedingungen erfüllt sind.

```
if ((meineMaus.hPosition() != 300 ) ||
                                meineMaus.istGedrueckt())
  meinStift.bewegeBis(20,30);
```

Der Stift wird im obigen Beispiel nur bewegt, wenn eine oder beide Bedingungen erfüllt sind.

Aufgaben

43_1 Der Benutzer wird aufgefordert, eine ganze Zahl einzugeben. Falls nicht die Zahl 1 eingegeben wurde, wird der Bildschirmhintergrund blau gefärbt.

43_2 Der Benutzer wird aufgefordert, eine ganze Zahl einzugeben. Falls die Zahl 1 oder die Zahl 5 eingegeben wurde, wird der Bildschirmhintergrund grün gefärbt.

43_3 Der Benutzer wird aufgefordert, eine Zahl einzugeben. Falls die Zahl zwischen 100 und 1000 liegt, wird der Bildschirmhintergrund schwarz gefärbt.

43_4 Der Benutzer wird aufgefordert, eine Zahl einzugeben. Falls die Zahl zwischen 100 und 200 oder zwischen 800 und 900 liegt, wird der Bildschirmhintergrund rosa gefärbt.

Die *Switch-Case*-Anweisung

```
.............
public void fuehreAus()
{
  meinStift.bewegeBis(50,50);

  int n = derBildschirm.holeGanzeZahl();
  switch (n)
  {
  case 1: meinStift.schreibeText("sehr gut"); break;
  case 2: meinStift.schreibeText("gut"); break;
  case 3:
  case 4: meinStift.schreibeText("O.K."); break;
  case 5: meinStift.schreibeText("mangelhaft"); break;
  case 6: meinStift.schreibeText("ungenügend"); break;
  default: meinStift.schreibeText("unbekannt");
  }
  // Aufräumen
  .....
  }
.............
```

Die runden Klammern hinter dem Wort *switch* (deutsch: Schalter) enthalten den Wert, der getestet wird. Als Datentyp für diesen sog. Selektor kommen nur sog. primitive Typen in Frage, bei denen man exakt auf Gleichheit prüfen kann. Bei Rechnungen mit dem Zahlentyp *double* kann z.B. aufgrund von mathematischen Rundungen nicht immer exakt gerechnet werden. Deshalb darf der Typ *double* in der switch-Anweisung nicht benutzt werden.
Auch *Objekte* dürfen nicht als Selektor benutzt werden.
Hinweis:
ab der Java-Version 1.7 darf auch der Datentyp *String* als Selektor genutzt werden. In manchen BlueJ-Versionen gibt es im BlueJ-Verzeichnis ein Hilfsprogramm namens *selectVM*, mit dem man die Java-Version auswählen kann, falls der Computer mehrere Java-Versionen installiert hat. Ansonsten lässt sich auch im Windows-System eine (von mehreren) Java-Versionen als Standard einstellen.

Nach den Doppelpunkten in der switch-case-Anweisung folgen die Anweisungen, die im entsprechenden Fall ausgeführt werden sollen. Interessant: Geschweifte Klammern werden hier nicht benötigt.

Problematisch ist das Wort *break*. Falls es vergessen wird, kommt keine Fehlermeldung, stattdessen werden die Anweisungen hinter dem nächsten *case* ausgeführt und zwar solange, bis ein *break* erreicht wird oder die switch-case-Anweisung zu Ende ist.
Dies hat im obigen Programmbeispiel zur Folge, dass auch bei Eingabe der Notenzahl 3 die Textausgabe `"O.K."` erfolgt.

Die Anweisungen hinter *default* (deutsch: Standardwert) werden in allen Fällen ausgeführt, die vorher nicht mit *case* abgefangen wurden. Die *default*-Zeile kann auch entfallen, dann passiert in den nicht vorher ausgewählten Fällen nichts.

```
.....
public void fuehreAus()
{
 // Aktionsteil
 meinStift.bewegeBis(50,50);

 String name = derBildschirm.holeText();
 switch (name)
 {
  case "Anton": meinStift.schreibeText("männl."); break;
  case "Julia": meinStift.schreibeText("weibl."); break;
  default: meinStift.schreibeText("unbekannt");
 }

 // Aufräumen
 .....
}
.....
```

Aufgaben

46_1 Der Benutzer wird aufgefordert, eine Zahl einzugeben. Bei Eingabe von 1 wird der Bildschirmhintergrund rot gefärbt, bei Eingabe von 2 gelb, ansonsten rosa.

46_2 Der Benutzer gibt den Namen eines Wochentages ein. Der Computer antwortet mit „Werktag“ oder „Wochenende“.

46_3 Der Benutzer gibt eine Monatszahl ein (also 1 bis 12). Der Computer antwortet mit dem entsprechenden Monatsnamen.

46_4 Der Benutzer gibt einen Monatsnamen ein. Der Computer antwortet mit der entsprechenden Monatszahl (also 1 bis 12).

Schleifen

Die sog. *while*-Schleife wird üblicherweise dann benutzt, wenn man nicht genau weiß, wie oft etwas gemacht werden soll. Die Bedingung wird gleich zu Beginn der Schleife geprüft. Falls diese Bedingung erfüllt sein sollte, wird die Anweisung einmal ausgeführt. Danach wird erneut geprüft usw.
Die *while*-Schleife wird eventuell auch garnicht ausgeführt!

```
while (meineMaus.istGedrueckt())
  meinStift.bewegeBis(meineMaus.hPosition(),
                                  meineMaus.vPosition());
meinStift.zeichneKreis(10);
```
.....

Bei einer ***while***-Schleife wird zuerst die **Eingang**sbedingung geprüft. Falls diese Bedingung erfüllt sein sollte, wird der Anweisungsteil einmal ausgeführt. Danach wird wieder die Eingangsbedingung überprüft usw.
Ist die Eingangsbedingung nicht erfüllt, wird die *while*-Schleife verlassen und es wird zum nächsten Befehl übergegangen**. Es kann also durchaus sein, dass der Anweisungsteil einer *while*-Schleife überhaupt nicht ausgeführt wird!**

Im folgenden Beispiel wartet das Programm solange, bis die Maus gedrückt wird. Entscheidend dafür ist u.a. das Semikolon. Da zwischen der Bedingung und dem Semikolon keine Anweisung steht, wird auch nichts gemacht:
.....
```
while (!meineMaus.istGedrueckt()) ;
```
.....

Sollte der Anweisungsteil bei einer *while*-Schleife aus mehr als einem Befehl bestehen, so müssen diese Befehle in geschweiften Klammern eingeschlossen werden:
```
while (meineMaus.istGedrueckt())
 {
  meinStift.bewegeBis(meineMaus.hPosition(),
                                  meineMaus.vPosition());
  meinStift.zeichneKreis(1);
 }
```
.....

Im folgenden Beispiel wird ein 5-Eck gezeichnet.

```
meinStift.bewegeBis(50,50);
meinStift.runter();
int i = 1;
while (i <= 5)
  {
   meinStift.bewegeUm(30);
   meinStift.dreheUm(72);
   i = i+1;
  }
```

Die sog. *do-while*-Schleife wird üblicherweise dann benutzt, wenn etwas mindestens einmal gemacht werden muss, aber nicht ganz klar ist, wie oft insgesamt.
Bei der ***do-while***-Schleife wird die Bedingung erst am Ende der Schleife geprüft. Falls diese Bedingung dann erfüllt sein sollte, wird die Anweisung noch einmal ausgeführt usw.
Die *do-while*-Schleife wird also mindestens einmal ausgeführt!

```
do
  {
   meinStift.bewegeUm(5);
   meinStift.zeichneKreis(1);
  }
while (meinStift.hPosition() <= 100);
```

Diese Technik werden wir in Zukunft oft benutzen, um ein Programm solange auszuführen, bis es mit einem Doppelklick der Maus beendet wird.

```
do
  {
   meinStift.bewegeBis(meineMaus.hPosition(),
                                meineMaus.vPosition());
   meinStift.zeichneRechteck(1,1);
  }
while (!meineMaus.doppelKlick());
```

Die sog. *for*-Schleife wird üblicherweise dann benutzt, wenn man genau weiß, wie oft etwas gemacht werden soll.

```
meinStift.bewegeBis(100,100);
meinStift.runter();
for (int i = 1; i <= 4; i++)
   {
     meinStift.bewegeUm(50);
     meinStift.dreheUm(90);
   }
```

```
meinStift.bewegeBis(100,100);
meinStift.runter();
for (int j = 3; j > 0; j--)
   {
     meinStift.bewegeUm(50);
     meinStift.dreheUm(120);
   }
```

```
meinStift.bewegeBis(10, 20);
for (int i = 4; i <= 40; i = i+4)
   {
     meinStift.schreibeZahl(i*i);
     meinStift.schreibeText("   ");
   }
```

Aufgaben

50_1 Zeichne mit jeder der drei Schleifenformen (while-, do-while-, for-Schleife) a) ein reguläres 9-Eck! b) ein reguläres 12-Eck!

50_2 Gib mit jeder der drei Schleifenformen auf dem Bildschirm die ersten 10 Quadratzahlen aus!

50_3. Zeichne eine analoge Uhr mit Stunden- und Minutenzeiger! Es wird die Zeit $5.^{12}$ Uhr angezeigt. Am Kreisrand der Uhr befindet sich jeweils ein kleiner Strich für jede Stunde.

Aufgabe 50_3

```
public void fuehreAus()
 {
   meinStift.bewegeBis(200, 200);   //Mittelpunkt
   meinStift.zeichneKreis(50);
   meinStift.hoch();
   meinStift.dreheBis(0);
   for (int i = 1; i <= 12; i++)
    {
      meinStift.dreheUm(30);
      meinStift.bewegeBis(200, 200);
      meinStift.hoch();
      meinStift.bewegeUm(42);
      meinStift.runter();
      meinStift.bewegeUm(5);
      meinStift.hoch();
    }

    meinStift.bewegeBis(200, 200);
    meinStift.dreheBis(18);    //Minutenzeiger
    meinStift.runter();
    meinStift.bewegeUm(37);
    meinStift.bewegeUm(-37);

    meinStift.dreheBis(-66);    //Stundenzeiger
    meinStift.bewegeUm(23);
    meinStift.bewegeUm(-23);
    meinStift.hoch();
 }
```

Freihandzeichnen

Die Programme 1 bis 5 sollen durch einen Doppelklick der Maus beendet werden.

52_1 Eine Linie auf dem Bildschirm folgt den Bewegungen der Maus. Kannst du dein Programm anschließend so verbessern, dass zu Beginn keine Linie von der oberen linken Ecke bis zur ersten Mausposition gezogen wird?

52_2 Bei gedrückter Maustaste wird permanent ein Punkt an der Mausposition gezeichnet. Drücke die Maustaste und bewege die Maus anschließend schnell eine kurze Strecke über den Bildschirm. Sind an der Mausspur die Beschleunigungen dieser Bewegung zu erkennen?

52_3 Mit einem sog. ***Klick*** (= Drücken **und** wieder loslassen) auf die Maustaste wird ein Punkt an der Mausposition gezeichnet.

52_4 (Freies Schreiben auf dem Bildschirm)
Solange die Maustaste gedrückt ist, folgt eine (gekrümmte) Linie auf dem Bildschirm den Bewegungen der Maus.

52_5 Wenn die Maustaste gedrückt wird, wird nur ein einziger Punkt an der betreffenden Mausposition gezeichnet. Wenn die Maustaste losgelassen wird, wird eine gerade Linie von diesem Punkt bis zur neuen Mausposition gezeichnet.

52_6 Implementiere die Methode
zeichneRegulaeresVieleck(double px, double py, double pk; int pn)
Dabei bedeuten *px* und *py* die Koordinaten eines Eckpunktes, *pk* die Kantenlänge und *pn* die Anzahl der Eckpunkte.

Lösungen

Aufgabe 52_1

```
import sum.kern.*;
…….
public class aufgabe1
{
     // Objekte
     Bildschirm derBildschirm;
     Stift meinStift;
     Maus meineMaus;

     // Konstruktor
     public aufgabe1()
     {
          derBildschirm = new Bildschirm(500,300);
          meinStift = new Stift();
          meineMaus = new Maus();
     }

     // Dienste
     public void fuehreAus()
     {
          meinStift.bewegeBis(meineMaus.hPosition(),
                              meineMaus.vPosition());
          meinStift.runter();
          do
            meinStift.bewegeBis(meineMaus.hPosition(),
                              meineMaus.vPosition());
          while (!meineMaus.doppelKlick());

          // Aufräumen
          ……..
     }
}
```

Aufgabe 52_2

```
……..
public void fuehreAus()
  {
   do
    {
     meinStift.bewegeBis(meineMaus.hPosition(),
                                 meineMaus.vPosition());
     if (meineMaus.istGedrueckt())
       meinStift.zeichneKreis(1);
    }
   while (!meineMaus.doppelKlick());

   // Aufräumen  ……..
  }
```

Aufgabe 52_3

```
……
public void fuehreAus()
  {
   do
    {
     if (meineMaus.istGedrueckt())
      {
       meinStift.bewegeBis(meineMaus.hPosition(),
                                 meineMaus.vPosition());
       meinStift.zeichneKreis(1);
      }
     while (meineMaus.istGedrueckt()) ;
    }
   while (!meineMaus.doppelKlick());

  }…….
```

Aufgabe 52_4

Lösungsversion a)

```
.....
public void fuehreAus()
  {
   do
    {
     meinStift.runter();
     while (meineMaus.istGedrueckt())
       meinStift.bewegeBis(meineMaus.hPosition(),
                           meineMaus.vPosition());
     meinStift.hoch();
     while (!meineMaus.istGedrueckt())
       meinStift.bewegeBis(meineMaus.hPosition(),
                           meineMaus.vPosition());
    }
   while (!meineMaus.doppelKlick());
  ............
  }
```

Lösungsversion b)

```
.............
public void fuehreAus()
  {
   do
    {
     if (meineMaus.istGedrueckt())
                                  meinStift.runter();
     else meinStift.hoch();
     meinStift.bewegeBis(meineMaus.hPosition(),
                         meineMaus.vPosition());
    }
   while (!meineMaus.doppelKlick());

  }
```

Aufgabe 52_5

…….

```
public void fuehreAus()
 {
  do
   {
    if (meineMaus.istGedrueckt())
     {
      meinStift.bewegeBis(meineMaus.hPosition(),
                                meineMaus.vPosition());
      meinStift.zeichneKreis(1);
      while (meineMaus.istGedrueckt()) ;
      meinStift.runter();
      meinStift.bewegeBis(meineMaus.hPosition(),
                                meineMaus.vPosition());
      meinStift.hoch();
     }
   }
  while (!meineMaus.doppelKlick());

 }
```

……

Die Klasse *Tastatur*

Tastatur
+ ! init() + ! gibFrei() + ! weiter() + ? wurdeGedrueckt() + ? zeichen()

Eine Tastatur realisiert die Zeicheneingabe des Computers. Sie speichert die eingegebenen Tastaturzeichen in der Reihenfolge ihrer Eingabe in einem eigenen Pufferspeicher.

Es gibt eine Klasse namens ***Zeichen***, welche folgende Konstanten als Tastatureingabe zur Verfügung stellt:
ESCAPE, ENDE, POS1, PFEILLINKS, PFEILRECHTS, PFEILOBEN, PFEILUNTEN, BILDUNTEN, BILDAUF, TAB, EINGABE, BACKSPACE, DELETE, F1, F2, ... , F12
Der Aufruf einer solchen Konstanten erfolgt z.B. durch
Zeichen.PFEILLINKS;

Konstruktor:

Tastatur()
Die Tastatur wird initialisiert und der Tastaturpuffer enthält keine Zeichen.

Methoden:

void **gibFrei**()
Die Tastatur steht nicht mehr zur Verfügung.

void **weiter**()
Das vorderste Zeichen im Tastaturpuffer wird entfernt. Falls der Tastaturpuffer vorher nicht mit "wurdeGedrueckt()" getestet wurde, erfolgt eine Fehlermeldung.

boolean **wurdeGedrueckt**()
Falls eine Taste gedrückt wurde, der Tastaturpuffer also (mindestens) ein Zeichen enthält, ist "wurdeGedrueckt" wahr, sonst falsch.
Hinweis: der Zustand *(wurdeGedrueckt() = TRUE)* ändert sich erst wieder, wenn der Auftrag *weiter()* (entsprechend oft) erteilt wird.

char **zeichen**()
Diese Anfrage liefert eine Kopie des zuerst eingegebenen Zeichens im Tastaturpuffer. Das Zeichen im Tastaturpuffer selbst wird dabei nicht gelöscht. Falls der Tastaturpuffer vorher nicht mit "wurdeGedrueckt()" gestestet wurde, erfolgt eine Fehlermeldung.

Im folgenden Anwendungsbeispiel wird die Tastatureingabe abgefragt. Hierbei gibt es mehrere Probleme: Zunächst muss das Programmfenster den Fokus erhalten, weil die Tastatureingaben vom System aus an das aktuelle Fenster geschickt werden. Wegen der schnellen Abfrage wird außerdem bei der Eingabe von Großbuchstaben erst mehrmals die Shift-Taste als Eingabe erkannt.

```
public void fuehreAus()
 {
  char ch;
  meinStift.bewegeBis(50,50);

  do
  {
    if (meineTastatur.wurdeGedrueckt())
     {
      ch = meineTastatur.zeichen();
      switch (ch)
       {
        case 'A': derBildschirm.loescheAlles();
                  meinStift.bewegeBis(50,50);
                  meinStift.schreibeText("Anton");
                                                  break;
        case Zeichen.PFEILRECHTS:
                  derBildschirm.loescheAlles();
                  meinStift.bewegeBis(50,50);
                  meinStift.schreibeText("Pfeilrechts");
                                                  break;
        default: derBildschirm.loescheAlles();
                  meinStift.bewegeBis(50,50);
                  meinStift.schreibeText("unbekannt");
      }
     meineTastatur.weiter();
    }
  }
  while (!meineMaus.doppelKlick());
.....
 }
```

Aufgaben

59_1 Der Rechner wartet auf Tastatureingaben. Es werden nur die 4 Pfeiltasten akzeptiert. Wird die Pfeilrechts-Taste gedrückt, so erscheint eine 50 Pixel lange Linie nach rechts. Analog wird bei den anderen Pfeiltasten reagiert. Das ganze wird solange durchgeführt, bis ein Mausdoppelklick erfolgt.

59_2 Wie Aufgabe 1. Zusätzlich werden die Ziffer 1 und die Buchstaben h, r und v akzeptiert. Bei Eingabe der Ziffer 1 dreht sich der Stift um 10 Grad nach links, die Buchstaben h und r sorgen dafür, dass der Stift hoch gehoben bzw. runter gelassen wird. Der Buchstabe v bewirkt ein Vorwärtsgehen des Stiftes um 50 Pixel. Letzteres ist sinnvoll, falls der Winkel des Stiftes vorher geändert wurde.

59_3 Solange die Maus nicht gedrückt wurde, passiert gar nichts. Nach dem lang erwarteten Mausklick geschieht folgendes:
Der Stift geht zum Mausort. Solange die Tastatur nicht gedrückt wird, passiert gar nichts. Nach der erwarteten Tastatureingabe wird am Mausort ein entsprechendes Wort geschrieben.
Das ganze wird solange durchgeführt, bis ein Mausdoppelklick erfolgt.

59_4 Solange der Mausknopf gedrückt ist, folgt eine Linie auf dem Bildschirm den Bewegungen der Maus. Zusätzlich ist es möglich, durch Drücken einer Taste auf Radieren zu wechseln. Das Programm wird durch einen Doppelklick beendet. (Bemerkung: Wenn man zusätzlich abfragt, welche Taste gedrückt wurde, so kann man zwischen Normal- und Radiermodus hin und her wechseln).
Wahrscheinlich wird dein Programm unterschiedlich reagieren, je nachdem, ob bei gedrückter Maustaste die Tastatur betätigt wurde oder nicht. Kannst du das erklären?

Lösungen

Aufgabe 59_1

```
public void fuehreAus()
 {
  char ch;
  meinStift.bewegeBis(derBildschirm.breite()/2,
                              derBildschirm.hoehe()/2);
  meinStift.runter();
  do
   {
    if (meineTastatur.wurdeGedrueckt())
     {
      ch = meineTastatur.zeichen();
      meineTastatur.weiter();
      switch (ch)
       {
        case Zeichen.PFEILRECHTS:
                              meinStift.dreheBis(0);
                              meinStift.bewegeUm(50);
                              break;
        case Zeichen.PFEILLINKS:
                              meinStift.dreheBis(180);
                              meinStift.bewegeUm(50);
                              break;
        case Zeichen.PFEILOBEN:
                              meinStift.dreheBis(90);
                              meinStift.bewegeUm(50);
                              break;
        case Zeichen.PFEILUNTEN:
                              meinStift.dreheBis(270);
                              meinStift.bewegeUm(50);
                              break;
       } // Ende von switch
     } // Ende von if
   } // Ende von do
  while (!meineMaus.doppelKlick() );.....
```

Aufgabe 59_4

```
.............
public void fuehreAus()
 {
  do
   {
    meinStift.runter();
    /* Wahrscheinlich ist zu Beginn des Programms der Mausknopf noch nicht
       gedrückt.
    */
    while (meineMaus.istGedrueckt())
      meinStift.bewegeBis(meineMaus.hPosition(),
                                  meineMaus.vPosition());
    if (meineTastatur.wurdeGedrueckt())
                                     meinStift.radiere();
    meinStift.hoch();
    while (!meineMaus.istGedrueckt())
      meinStift.bewegeBis(meineMaus.hPosition(),
                                  meineMaus.vPosition());
   }
  while (!meineMaus.doppelKlick());
.....................
 }
```

Die Klasse *Buntstift*

Oberklasse: *Stift*

Die Klasse Buntstift besitzt sämtliche Attribute und Methoden der Oberklasse Stift. (Das ist übrigens nicht selbstverständlich!). Allerdings besitzt sie darüber hinausgehende Attribute, die mithilfe von geeigneten Methoden einzeln gesetzt werden können.

Buntstift()
Der Buntstift wird als Stift initialisiert und mit den Standardeinstellungen versehen.

int **linienbreite**()
int **linienBreite**()
Der Buntstift liefert seine Linienbreite.

void **setzeFarbe**(int pFarbe)
Die angegebene Farbe wird die aktuelle Farbe des Buntstifts.
*Es sei hier an die bereits bekannte Klasse namens **Farbe** erinnert (vgl. S. 8 und 9 in diesem Skript!), welche als sog. Klassenkonstanten einige Standardfarben als int-Werte zur Verfügung stellt.*
Beispiel: meinBuntstift.setzeFarbe(Farbe.ROT);

void **setzeFuellmuster**(int pMuster)
void **setzeFuellMuster**(int pMuster)
Das angegebene Muster ist das aktuelle Muster des Buntstifts für Rechtecke und Kreise.
*Es gibt eine Klasse namens **Muster**, welche als Klassenkonstanten nur drei int-Werte als Standardfüllmuster zur Verfügung stellt, nämlich DURCHSICHTIG, DURCHSCHEINEND, GEFUELLT*
Beispiel: meinBuntstift.setzeFuellmuster(Muster.DURCHSCHEINEND);

void **setzeLinienbreite**(int pBreite)
void **setzeLinienBreite**(int pBreite)
Die angegebene Breite wird die aktuelle Linienbreite des Buntstifts.

void **setzeSchriftart**(String pArt)
void **setzeSchriftArt**(String pArt)
Die angegebene Schriftart wird die aktuelle Schriftart des Buntstifts. Allerdings muss die angegebene Schriftart auf dem Computer installiert sein.
*Es gibt eine Klasse namens **Schrift**, welche als Klassenkonstanten folgende vier Standardschriftarten als String-Werte zur Verfügung stellt: ARIAL, HELVETICA, STANDARDSCHRIFTART (entspricht HELVETICA), TIMESROMAN*
Beispiel: Folgende drei Anweisungen sind in ihrer Wirkung identisch:
meinBuntstift.setzeSchriftart(Schrift.HELVETICA);
meinBuntstift.setzeSchriftart("Helvetica");
meinBuntstift.setzeSchriftart(Schrift. STANDARDSCHRIFTART);

void **setzeSchriftgroesse**(int pGroesse)
void **setzeSchriftGroesse**(int pGroesse)
Die angegebene Schriftgroesse wird die aktuelle Schriftgröße des Buntstifts.

void **setzeSchriftstil**(int pStil)
void **setzeSchriftStil**(int pStil)
Der angegebene Schriftstil wird der aktuelle Schriftstil des Buntstifts.
*Es gibt eine Klasse namens **Schrift**, welche als Klassenkonstanten folgende vier Standardschriftstile als int-Werte zur Verfügung stellt: STANDARDSTIL, FETT, KURSIV, STANDARDGROESSE*
Beispiel: meinBuntstift.setzeSchriftstil(Schrift.FETT);

int **textbreite**(String pText)
int **textBreite**(String pText)
ermittelt die Breite des Textes unter Berücksichtigung der Eigenschaften des Buntstifts.

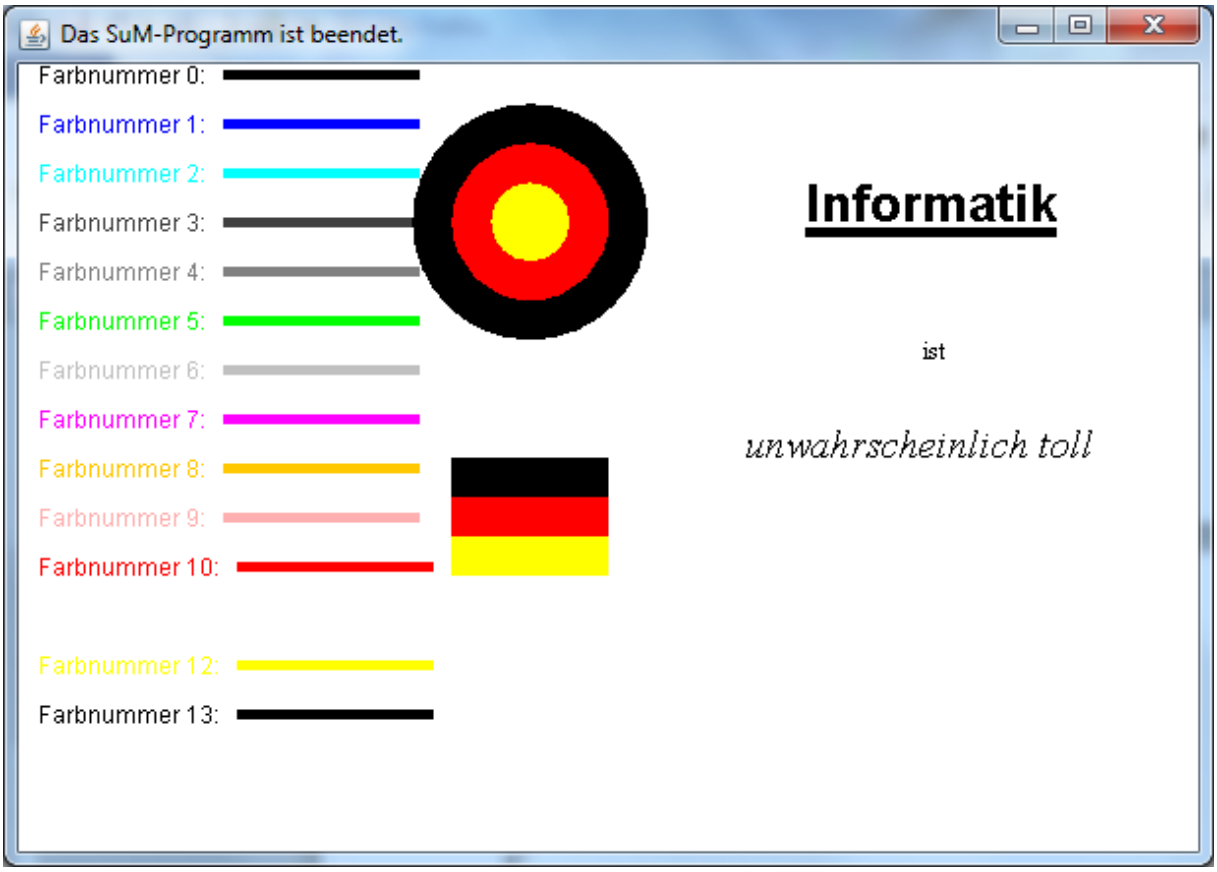
int **zahlbreite**(double pZahl)
int **zahlBreite**(double pZahl)
int **zahlBreite**(int pZahl)
ermittelt die Breite der Zahl unter Berücksichtigung der Eigenschaften des Buntstifts.

int **zeichenbreite**(char pZeichen)
int **zeichenBreite**(char pZeichen)
ermittelt die Breite des Zeichens unter Berücksichtigung der Eigenschaften des Buntstifts.

Stift
Position # Richtung # Zustand # Modus
+ ! init() + ! bewegeBis() + ! bewegeUm() + ! dreheBis() + ! dreheUm() + ! dreheZu() + ! gibFrei() + ! hoch() + ? hPosition() + ? istUnten() + ! normal() + ! radiere() + ! runter() + ! schreibeZahl() + ! schreibeText() + ? vPosition() + ! wechsle() + ? winkel() + ! zeichneKreis() + ! zeichneRechteck()

Buntstift
- Farbe - Schriftart - Schriftgröße - Linienbreite - Füllmuster
+ ! init() + ? linienbreite() + ! setzeFarbe() + ! setzeFuellmuster() + ! setzeLinienbreite() + ! setzeSchriftart() + ! setzeSchriftgroesse() + ! setzeSchriftstil() + ? textbreite() + ? zahlbreite() + ? zeichenbreite()

Programmiere das folgende Bild!



Lösung

```
import sum.kern.*;
public class aufgabe
{
 // Objekte
 Bildschirm derBildschirm;
 Buntstift meinBuntstift;
 Maus meineMaus;
 Tastatur meineTastatur;

 // Konstruktor
 public aufgabe()
  {
      derBildschirm = new Bildschirm(600,400);
      derBildschirm.nachVorn();
      meinBuntstift = new Buntstift();
      meineMaus = new Maus();
      meineTastatur = new Tastatur();
  }

  // Dienste
  void flagge()
   {
    meinBuntstift.setzeFuellMuster(Muster.GEFUELLT);
    meinBuntstift.bewegeBis(220,200);
    meinBuntstift.setzeFarbe(Farbe.SCHWARZ);
    meinBuntstift.zeichneRechteck(80,20);

    meinBuntstift.bewegeBis(220,220);
    meinBuntstift.setzeFarbe(Farbe.ROT);
    meinBuntstift.zeichneRechteck(80,20);

    meinBuntstift.bewegeBis(220,240);
    meinBuntstift.setzeFarbe(Farbe.GELB);
    meinBuntstift.zeichneRechteck(80,20);
   }
```

```
void zielscheibe()
 {
  meinBuntstift.setzeFuellMuster(Muster.GEFUELLT);

  meinBuntstift.bewegeBis(260,80);
  meinBuntstift.setzeFarbe(Farbe.SCHWARZ);
  meinBuntstift.zeichneKreis(60);

  meinBuntstift.bewegeBis(260,80);
  meinBuntstift.setzeFarbe(Farbe.ROT);
  meinBuntstift.zeichneKreis(40);

  meinBuntstift.bewegeBis(260,80);
  meinBuntstift.setzeFarbe(Farbe.GELB);
  meinBuntstift.zeichneKreis(20);
}


void farbNummern()
 {
  int x = 10;
  int y = 10;
  int f = 0;
  meinBuntstift.setzeLinienBreite(5);
  do
   {
    meinBuntstift.bewegeBis(x,y);
    meinBuntstift.setzeFarbe(f);
    meinBuntstift.schreibeText("Farbnummer ");
    meinBuntstift.schreibeZahl(f);
    meinBuntstift.schreibeText(":   ");
    meinBuntstift.bewegeBis(meinBuntstift.hPosition(),
                        meinBuntstift.vPosition()-5);
    meinBuntstift.runter();
    meinBuntstift.bewegeUm(100);
    meinBuntstift.hoch();
```

```
    y = y+25;
    f = f+1;
   }
   while (f <= 13);
 }


public void fuehreAus()
 {
  double b;

  farbNummern();
  flagge();
  zielscheibe();

  meinBuntstift.setzeSchriftArt(Schrift.ARIAL);
  meinBuntstift.setzeSchriftGroesse(26);
  meinBuntstift.setzeFarbe(0);
  meinBuntstift.bewegeBis(400, 80);
  meinBuntstift.setzeSchriftStil(Schrift.FETT);
  meinBuntstift.schreibeText("Informatik");
  b = meinBuntstift.textbreite("Informatik");
  meinBuntstift.bewegeBis(400, 80+5);
  meinBuntstift.runter();
  meinBuntstift.bewegeUm(b);
  meinBuntstift.hoch();

  meinBuntstift.setzeSchriftArt(Schrift.TIMESROMAN);
  meinBuntstift.setzeSchriftGroesse(12);
  meinBuntstift.bewegeBis(460, 150);

  meinBuntstift.setzeSchriftStil(Schrift.STANDARDSTIL);
  meinBuntstift.schreibeText("ist");
  meinBuntstift.hoch();


  meinBuntstift.setzeSchriftArt("Times New Roman");
  meinBuntstift.setzeSchriftGroesse(20);
```

```
    meinBuntstift.setzeSchriftStil(Schrift.KURSIV);
    meinBuntstift.bewegeBis(370, 200);
    meinBuntstift.schreibeText("unwahrscheinlich toll");

    // Aufräumen
    meineTastatur.gibFrei();
    meineMaus.gibFrei();
    meinBuntstift.gibFrei();
    derBildschirm.gibFrei();
  }
}
```

Aufgaben

70_1 Untersuche, welche Schriftstile sich hinter den Zahlen 0 bis 12 verbergen!

70_2 Untersuche, ob sich beliebige Schriftarten (z.B. "Wingdings") erzeugen lassen!

70_3 Es soll ein Haus gezeichnet werden. Die Außenmauern sind breiter als z.B. die Fensterrahmen. Fenster und Tür haben natürlich unterschiedliche Farben**.** Schreibe ein passendes Programm!

70_4 Wie Aufgabe 3. Schreibe diesmal eine Prozedur mit Parametern, z.B. *procedure Haus(double x, double y, double breite, double hoehe, int f)* so dass man sehr einfach unterschiedlich große Häuser an unterschiedlichen Orten mit unterschiedlicher Grundfarbe erzeugen kann.

70_5 Wird der Mausknopf gedrückt, so wird nur ein Punkt an der Mausposition gezeichnet. Wenn der Mausknopf losgelassen wird, wird eine gerade Linie von diesem Punkt bis zur neuen Mausposition gezeichnet. Das Programm wird durch einen Doppelklick beendet. Zusätzlich ist es möglich, durch Drücken einer beliebigen Taste die Zeichenfarbe auf rot zu wechseln.

70_6 Wie Aufgabe 5, aber der Farbwechsel geschieht durch Drücken der Taste „r“ (für rot) bzw. „g“ (für gelb).

70_7 Wie Aufgabe 5. Es ist möglich, durch Drücken einzelner Buchstaben die Zeichenfarbe zu wechseln, einen Kreis oder ein Rechteck zu zeichnen, die Linienbreite einzustellen usw. .

Lösungen

Aufgabe 70_1

```
void stile()
 {
  int x = 10;
  int y = 20;
  int s = 0;
  meinBuntstift.setzeSchriftGroesse(20);
  do
   {
    meinBuntstift.bewegeBis(x,y);
    meinBuntstift.setzeSchriftStil(s);
    meinBuntstift.schreibeText("Stilnummer "+s+ " ");
    meinBuntstift.schreibeText("Das Goethe-Gymnasium
                                          ist toll");
    meinBuntstift.hoch();

    meinBuntstift.bewegeBis(meinBuntstift.hPosition(),
                        meinBuntstift.vPosition()-5);

    y = y+25;
    s = s+1;
   }
   while (s <= 12);
 }
```

Aufgabe 70_2

```
void arten() {
  meinBuntstift.setzeSchriftGroesse(20);
  meinBuntstift.bewegeBis(10,20);
  meinBuntstift.setzeSchriftart("Harlow Solid Italic");
  meinBuntstift.schreibeText("Schule ist toll");
}
```

Aufgabe 70_5

```
public void fuehreAus()
 {
  do
  {
   if (meineMaus.istGedrueckt())
    {
     meinBuntstift.bewegeBis(meineMaus.hPosition(),
                             meineMaus.vPosition());
     meinBuntstift.zeichneKreis(1);
     while (meineMaus.istGedrueckt()) ;
     meinBuntstift.runter();
     meinBuntstift.bewegeBis(meineMaus.hPosition(),
                             meineMaus.vPosition());
     meinBuntstift.hoch();
    }
   if (meineTastatur.wurdeGedrueckt())
    {
     if (meineTastatur.zeichen() == 'r')
       meinBuntstift.setzeFarbe(Farbe.ROT);
     if (meineTastatur.zeichen() == 'g')
       meinBuntstift.setzeFarbe(Farbe.GELB);
     meineTastatur.weiter();
    }
  } while (!meineMaus.doppelKlick());

        // Aufräumen
        meineTastatur.gibFrei();
        meineMaus.gibFrei();
        meinBuntstift.gibFrei();
        derBildschirm.gibFrei();
 }
```

Aufgabe 70_6

```
// wie in Aufgabe 5, neu ist:
if (meineTastatur.wurdeGedrueckt())
 {
  switch (meineTastatur.zeichen())
   {
    case 'r': meinBuntstift.setzeFarbe(Farbe.ROT);
                                                  break;
    case 'g': meinBuntstift.setzeFarbe(Farbe.GELB);
                                                  break;
    case 'b': meinBuntstift.setzeFarbe(Farbe.BLAU);
                                                  break;
    case 's': meinBuntstift.setzeFarbe(Farbe.SCHWARZ);
                                                   break;
    case 'K': meinBuntstift.hoch();
           meinBuntstift.bewegeBis(meineMaus.hPosition(),
                                   meineMaus.vPosition());
            meinBuntstift.zeichneKreis(10); break;
    case 'R': meinBuntstift.hoch();
       meinBuntstift.bewegeBis(meineMaus.hPosition(),
                                   meineMaus.vPosition());
              meinBuntstift.zeichneRechteck(20, 10);
                                                   break;
    case '3': meinBuntstift.setzeLinienBreite(3);
   } // of CASE
       meineTastatur.weiter();
 }.....
```

Die Klasse *Uhr*

Die Klasse Uhr ist in dem Paket *sum.werkzeuge* enthalten. Im Java-Programm muss also zu Beginn auch (zusätzlich) dieses Paket importiert werden:
*import sum.werkzeuge.**

Uhr() Eine Uhr wird initialisiert.

String **datum()**
Es wird das aktuelle Datum als Zeichenkette zurückgegeben.

double **gestoppteZeit()**
Die zuletzt gestoppte Zeit in Millisekunden wird geliefert.

void **gibFrei()**
Dummy-Prozedur.

int **jahr()**
gibt das aktuelle Jahr zurück.

int **minute()**
gibt die aktuelle Minute zurück.

int **monat()**
gibt die aktuelle Monatszahl (1 – 12) zurück.

int **sekunde()**
gibt die aktuelle Sekunde zurück.

void **starte()**
Die Stoppuhr wurde gestartet.

void **stoppe()**
Die Stoppuhr wurde gestoppt.

int **stunde()**
Die aktuelle Stunde wird zurückgegeben.

int **tag()**
Der aktuelle Tag wird zurückgegeben.

double **verstricheneZeit()**
Die seit Erzeugung der Uhr bzw. die seit dem letzten Start verstrichene Zeit wird (auf Tausendstel genau) geliefert.

void **warte(long pDauer)**
Es wird eine Pause von *pDauer* Millisekunden aufgerufen.

String **zeit()**
Es wird die aktuelle Zeit als Zeichenkette zurückgegeben.

Aufgaben

75_1 Gib das aktuelle Datum und die aktuelle Uhrzeit auf dem Bildschirm aus!

75_2 Versuche, ein Zeitintervall von 10 Sekunden abzuschätzen, indem du die Stoppuhr mit Mausklick startest und stoppst. Gib die gestoppte Zeit (in ms) auf dem Bildschirm aus!
Hinweis: Wenn man die Maus auch nur kurz drückt, so ist sie trotzdem eine Zeitlang im gedrückten Zustand. Das ist bei der Programmierung zu beachten. Man muss also softwaremäßig sicherstellen, dass die Maus zwischendurch wieder nicht gedrückt ist, bevor man das zweitemal gedrückt sein (als Stoppzeichen) abfragt.

75_3 Programmiere eine Digitaluhr, welche im Sekundentakt die Zeit anzeigt!
Hinweis: Speichere die Zeit in einer Variablen, gib sie aus, warte eine Sekunde lang, lösche sie anschließend mit Hilfe des Radiermodus bevor du die nächste Zeit ausgibst.

75_4 Programmiere nebenstehende analoge (und digitale) Uhr, welche im Sekundentakt die Zeit anzeigt!

17:41:05

Lösungen

Aufgabe 75_2

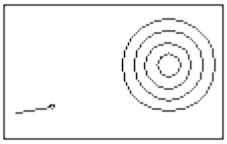
```
public void fuehreAus()
 {
  while (!meineMaus.istGedrueckt());
  meineUhr.starte();
  while (meineMaus.istGedrueckt());
  while (!meineMaus.istGedrueckt());
  meineUhr.stoppe();
  double zeit = meineUhr.gestoppteZeit();

  meinStift.bewegeBis(10,20);
  meinStift.schreibeZahl(zeit);

  while (!meineMaus.doppelKlick());

. // Aufräumen
    }
```

Aufgabe 75_3

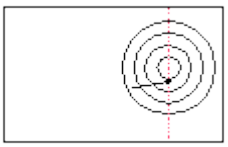
```
private void digitalUhr()  {
  String time;
  do  {
    meinStift.normal();
    meinStift.bewegeBis(10,20);
    time = meineUhr.zeit();
    meinStift.schreibeText(time);
    meineUhr.warte(1000);
    meinStift.radiere();
    meinStift.bewegeBis(10,20);
    meinStift.schreibeText(time);
   }
   while(!meineMaus.doppelKlick());
}
```

Projekt Pfeilwurf

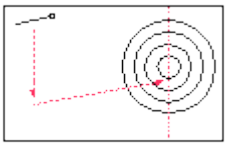
1. **Pfeil und Scheibe**
   Zeichne mit Hilfe des Stifts eine Dartscheibe und einen Pfeil gemäß nebenstehender Abbildung.



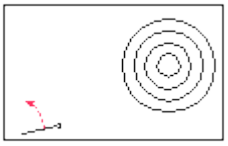
2. **Pfeil fliegt**
   Nun soll der Pfeil zur Dartscheibe fliegen. An einer gedachten Linie bleibt der Pfeil stehen.



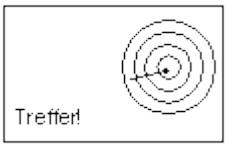
3. **Pfeil fällt**
   Der Pfeil fällt zunächst von oben senkrecht nach unten. Mit einem Mausdruck wird der Pfeil abgefeuert.



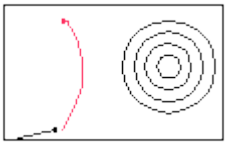
4. **Pfeil dreht**
   Nachdem der Pfeil gefallen ist, kann man noch die Wurfrichtung ändern: Während der Spieler die Maustaste gedrückt hält, soll sich der Pfeil und damit die Wurfrichtung (entgegen dem Uhrzeigersinn) ändern.



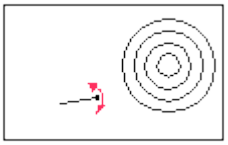
5. **Treffer?**
   Es wird analysiert, ob der Pfeil das Zentrum der Dartscheibe getroffen hat.



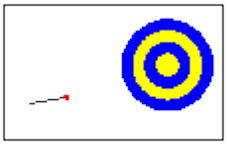
6. **Wieder hoch**
   Wenn der Spieler nicht aufpasst, kann der Pfeil aus dem unteren Bildschirmrand herauslaufen. Das soll nun verbessert werden. Wenn der Pfeil über den unteren Bildschirmrand läuft, soll er wieder nach oben versetzt werden.



7. **Richtungskorrektur**
   Mit einem Mausdruck wird der Pfeil innerhalb des freien Falls abgeschossen. Anschließend kann man noch während des Fluges mit den Tasten „r“ und „l“ die Wurfrichtung etwas nach rechts oder links ändern.



8. **Farbe**
   Das Spiel soll durch Einsatz von Farbe etwas aufgepeppt werden.



Lösungen

Aufgabe 2

```
import sum.kern.*;
import sum.werkzeuge.*; //enthält eine Klasse Uhr
……..
// Dienste
private void pfeilFlieg()
 {
  Uhr stoppuhr = new Uhr();
  do
   {
    meinStift.radiere();
    zeichnePfeil();
    meinStift.bewegeUm(1);
    meinStift.normal();
    zeichnePfeil();
    stoppuhr.warte(10); // ungefähre Anzahl in Millisekunden
   }
   while (meinStift.hPosition() <= 690);
 }


private void zeichnePfeil()
 {
  meinStift.runter();
  meinStift.bewegeUm(10);
  meinStift.dreheUm(150);
  meinStift.bewegeUm(3);
  meinStift.bewegeUm(-3);
  meinStift.dreheUm(-300);
  meinStift.bewegeUm(3);
  meinStift.bewegeUm(-3);
  meinStift.dreheUm(150);
  meinStift.bewegeUm(-10);
 }
```

```
private void zeichneZielscheibe()
 {
  meinStift.bewegeBis(700, 100);
  meinStift.zeichneKreis(65);
  meinStift.zeichneKreis(40);
  meinStift.zeichneKreis(15);
 }


public void fuehreAus()
   {
    zeichneZielscheibe();
    meinStift.bewegeBis(30,485);
    meinStift.dreheBis(30);
    pfeilFlieg();

    // Aufräumen
    ....................
   }
```

Die Klasse *Rechner*

Das Package ***sum.werkzeuge*** enthält eine Klasse namens ***Rechner***. Hier werden nur wenige ihrer Dienste vorgestellt.

Konstruktor:

Rechner()
Der Rechner wird initialisiert.

Methoden:

int **ganzeZufallszahl**()
int **ganzeZufallsZahl**()

int **ganzeZufallszahl**(int pVon, int pBis)
int **ganzeZufallsZahl**(int pVon, int pBis)
Es wird eine ganze Zufallszahl zwischen pVon und pBis (einschließlich der Grenzen) zurückgegeben.

int **gerundet**(double pZahl)

double **quadrat**(double pZahl)
int **quadrat**(int pZahl)

double **cos**(double pWinkel)
double **sin**(double pWinkel)
double **tan**(double pWinkel)

double **wurzel**(double pZahl)

double **zufallszahl**()
double **zufallsZahl**()
Es wird eine Zufallszahl zwischen 0 und 1 zurückgegeben.

Aufgaben

81_1 An zufälligen Orten auf dem Bildschirm sollen 100 kleine Sterne gezeichnet werden.

81_2 Nach zufälligen Zeitintervallen sollen an zufälligen Orten 100 kleine Sterne erscheinen.

81_3 Ermittle folgendermaßen die Reaktionszeit: Gezeigt wird ein roter Bildschirm. Nach einer zufälligen Zeit zwischen 2 und 10 Sekunden wird der Bildschirm grün. Daraufhin soll möglichst schnell die Maus gedrückt werden. Die Reaktionszeit wird auf dem Bildschirm angegeben.

81_4 Es sollen 100 zufällig farbige Kreise an zufälligen Orten gezeichnet werden.

81_5 Es sollen mehrere, zufällig unterschiedlich große Häuser gezeichnet werden.

81_6 Es soll ein Zufallsweg („Zick-Zack-Weg“) gezeichnet werden. Startpunkt ist die Mitte des Bildschirms.

81_7 An zufälligen Orten auf dem Bildschirm sollen 100 zufällig große Kreise gezeichnet werden.

81_8 Gegeben sei ein rechtwinkliges Dreieck mit der Hypotenuse c. Der Benutzer gibt die Seiten a und c ein. Der Rechner berechnet daraufhin die Seite b und gibt das Ergebnis auf dem Bildschirm aus.

81_9 Gegeben sei ein rechtwinkliges Dreieck mit der Hypotenuse c. Der Benutzer gibt die Seite a und den Winkel α ein. Der Rechner berechnet daraufhin die Seiten b und c und den Winkel β und gibt die Ergebnisse auf dem Bildschirm aus.

82_10 Gegeben sei ein beliebiges Dreieck. Der Benutzer gibt die Seiten b und c und den Winkel α ein. Der Rechner berechnet daraufhin die Seite a und gibt das Ergebnis auf dem Bildschirm aus (benutze den Cosinus-Satz!).

82_11 Gegeben sei ein beliebiges Dreieck. Der Benutzer gibt die Seite b und die Winkel α und β ein. Der Rechner berechnet daraufhin die Seite a und gibt das Ergebnis auf dem Bildschirm aus (benutze den Sinus-Satz!).

Lösungen

```
import sum.kern.*;
import sum.werkzeuge.*;
…….
private void Aufgabe81_1()
 {
  int i = 1;
  while (i <= 100)
   {
    int x = meinRechner.ganzeZufallszahl(0,
                              derBildschirm.breite());
    int y = meinRechner.ganzeZufallszahl(0,
                               derBildschirm.hoehe());
    meinStift.bewegeBis(x,y);
    meinStift.schreibeText("*");
    i = i+1;
   }
 }

public void fuehreAus()
 {
  Aufgabe1();
  ……
```

```
private void Aufgabe81_6()
 {
  meinStift.bewegeBis(derBildschirm.breite()/2,
                              derBildschirm.hoehe()/2);
  meinStift.runter();
  int i = 1;
  while (i <= 100)
   {
    int winkel = meinRechner.ganzeZufallszahl(0,360);
    meinStift.dreheUm(winkel);
    int laenge = meinRechner.ganzeZufallszahl(1,40);
    meinStift.bewegeUm(laenge);
    i = i+1;
   }
 }
```

```
private void Aufgabe81_8()

 {
  double a = derBildschirm.holeZahl("Bitte die Seite
                                        a eingeben: ");
  double c = derBildschirm.holeZahl("Bitte die Seite
                                        c eingeben: ");
  double b =
          meinRechner.wurzel(meinRechner.quadrat(c) -
                               meinRechner.quadrat(a));
  meinStift.bewegeBis(100,100);
  meinStift.schreibeZahl(b);
 }
```

```
private void Aufgabe81_9()
 {
  double a = derBildschirm.holeZahl("Bitte die Seite
                                        a eingeben: ");
  double alpha = derBildschirm.holeZahl("Bitte den
                             Winkel alpha eingeben: ");
  double c = a/meinRechner.sin(alpha);
  meinStift.bewegeBis(100,100);
  meinStift.schreibeZahl(c);
  double b =
          meinRechner.wurzel(meinRechner.quadrat(c) -
                               meinRechner.quadrat(a));
  meinStift.bewegeBis(100,150);
  meinStift.schreibeZahl(b);
  double beta = 90 - alpha;
  meinStift.bewegeBis(100,200);
  meinStift.schreibeZahl(beta);
 }
```

```
private void Aufgabe82_10()
 {
  double b = derBildschirm.holeZahl("Bitte die Seite
                                        b eingeben: ");
  double c = derBildschirm.holeZahl("Bitte die Seite
                                        c eingeben: ");
  double alpha = derBildschirm.holeZahl("Bitte den
                             Winkel alpha eingeben: ");
  double a = meinRechner.wurzel( b*b + c*c -
                        2*b*c*meinRechner.cos(alpha));
  meinStift.bewegeBis(100,100);
  meinStift.schreibeZahl(a);
 }
```

```
private void Aufgabe82_11()
 {
  double b = derBildschirm.holeZahl("Bitte die Seite
                                        b eingeben: ");
  double alpha = derBildschirm.holeZahl("Bitte den
                             Winkel alpha eingeben: ");
  double beta = derBildschirm.holeZahl("Bitte den
                              Winkel beta eingeben: ");
  double a =
      b*meinRechner.sin(alpha)/meinRechner.sin(beta);
  meinStift.bewegeBis(100,100);
  meinStift.schreibeZahl(a);
 }
```

Entwicklung eigener Klassen

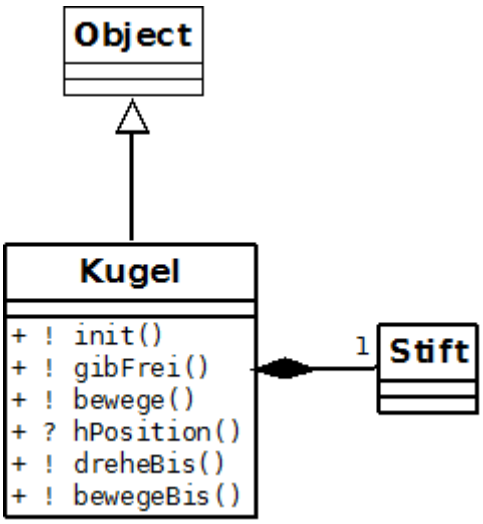
Aufgabe: Eine Kugel soll sich horizontal hin und her bewegen.
Lösung:

```
public void fuehreAus()
{
 meinStift.bewegeBis(derBildschirm.breite() / 2,
                          derBildschirm.hoehe() / 2);
 do
  {
   meinStift.radiere();
   meinStift.zeichneKreis(10);
   meinStift.hoch();
   meinStift.bewegeUm(1);
   meinStift.runter();
   meinStift.normal();
   meinStift.zeichneKreis(10);
   if (meinStift.hPosition() >=derBildschirm.breite()-10)
     meinStift.dreheBis(180);
   if (meinStift.hPosition() <= 10)
     meinStift.dreheBis(0);
  }
  while (!meineMaus.doppelKlick());

        // Aufräumen
  ..........
```

Wenn man nun mehrere, sich bewegende Kugeln auf dem Bildschirm haben möchte, so wäre der entsprechende Aufwand im Hauptprogramm ziemlich groß. Man benötigte eventuell mehrere Stifte zum Zeichnen der Kugeln. Deshalb ist es günstiger, man erstellt eine eigene Klasse namens *Kugel*. Alle Objekte dieser Klasse sollen sich selber bewegen und deshalb auch selber zeichnen können. Das Hauptprogramm sollte nur das Nötigste regulieren.

Wir werden im Folgenden eine zunächst einfache Klasse namens ***Kugel*** erstellen.

Die Klasse Kugel



Ein Objekt der Klasse Kugel hat einen eigenen Stift, mit dem sich das Objekt selber zeichnen kann. Bei einer ***Hat-Beziehung*** ist das besitzende Objekt verantwortlich für die Erzeugung und Freigabe des besessenen Objektes.

Erstelle nun folgendermaßen eine neue Klasse namens *Kugel*:

// Version 1

{ diese erste Version setzt noch voraus, dass ein Bildschirm bereits existiert, auf dem Exemplare der Klasse Kugel gezeichnet werden können. }

```
import sum.kern.*;

public class Kugel extends Object
```
// wenn der Zusatz extends Oberklasse *fehlt, wird die neue Klasse automatisch von der obersten Klasse namens* Object *abgeleitet.*
```
{
 // Bezugsobjekte
 private Stift hatStift;
 // Konstruktor; beachte, dass das Wort void fehlt!
 public Kugel()
 {
  hatStift = new Stift();
 }
 // Dienste
 public void bewege()
  {
   hatStift.radiere();
   hatStift.zeichneKreis(20);
   hatStift.hoch();
   hatStift.bewegeUm(0.1);
   hatStift.runter();
   hatStift.normal();
   hatStift.zeichneKreis(20);
  }

 public void bewegeBis(double px, double py)
  {
   hatStift.radiere();
   hatStift.zeichneKreis(20);
   hatStift.hoch();
   hatStift.bewegeBis(px, py);
   hatStift.runter();
   hatStift.normal();
   hatStift.zeichneKreis(20);
  }

 public void dreheBis(double pWinkel)
  {
```

```
    hatStift.dreheBis(pWinkel);
   }

 public void gibFrei()
   {
    hatStift.radiere();
    hatStift.zeichneKreis(20);
    hatStift.gibFrei();
   }

 public double hPosition()
   {
    return hatStift.hPosition();
   }
}
```

Das zugehörige Hauptprogramm lautet nun:

```
import sum.kern.*;
……
public class aufgabe
{
 // Objekte
 Bildschirm derBildschirm;
 Maus meineMaus;
 Kugel meineKugel;

 // Konstruktor
 public aufgabe()
 {
   derBildschirm = new Bildschirm(800,500);
   meineMaus = new Maus();
   meineKugel = new Kugel();
 }

 // Dienste
 public void fuehreAus()
 {
  meineKugel.bewegeBis(derBildschirm.breite() / 2,
                       derBildschirm.hoehe() /2);
  do
   {
    meineKugel.bewege();
    if(meineKugel.hPosition()>=derBildschirm.breite()-10)
      meineKugel.dreheBis(180);
    if (meineKugel.hPosition() <= 10)
      meineKugel.dreheBis(0);
   }
   while (!meineMaus.doppelKlick());
  // Aufräumen
  meineKugel.gibFrei();
  meineMaus.gibFrei();
  derBildschirm.gibFrei();
 }
}
```

Aufgaben

Alle folgenden Programme sollen mit einem Mausdoppelklick beendet werden können!

92_1 Schreibe eine Dokumentation der Klasse ***Kugel***!

92_2 Erzeuge zwei Kugeln, die sich örtlich übereinander auf dem Bildschirm horizontal hin und her bewegen!

92_3 Erzeuge zwei Kugeln, die sich örtlich übereinander auf dem Bildschirm horizontal hin und her bewegen! Eine Kugel soll sich doppelt so schnell bewegen wie die andere.

92_4 Erzeuge zwei Kugeln, die sich auf gleicher Höhe, horizontal und gleich schnell hintereinander her bewegen!

92_5 Die Klasse *Kugel* soll nun um einige Methoden erweitert werden:
- Schreibe die Funktion ***public double vPosition()***
- Um die Ballgeschwindigkeit einfach bestimmen zu können, ändere die Prozedur ***bewege()*** um in ***bewegeUm(double v)***
- Schreibe eine Prozedur namens ***public setzeRadius(int zahl)***

92_6 Eine einzige Kugel soll sich auf dem Bildschirm vertikal hin und her bewegen!

92_7 Eine einzige Kugel soll sich schräg über den Bildschirm bewegen. An allen 4 Bildschirmrändern wird sie reflektiert.

Bei mehr als zwei Kugeln wird der Aufwand im Hauptprogramm immer größer. Wenn sich die Kugeln auch noch gegenseitig abstoßen sollten, würde das Hauptprogramm immer umfangreicher.

Ein wichtiges Ziel in der Informatik ist es, das Hauptprogramm möglichst übersichtlich zu gestalten. Dazu ist es notwendig, möglichst viel *„Intelligenz“* in die untergeordneten Objekte zu stecken. In unserem Beispiel müssten die Kugeln selbst erkennen, wann sie in die Nähe eines Bildschirmrandes oder einer anderen Kugel kommen, um entsprechend reagieren zu können. Dafür müssten sie aber erst einmal die Größe des Bildschirmes und die anderen Kugeln ***kennen***.
Wir werden später eine verbesserte Version der Klasse *Kugel* realisieren.

Lösungen

Aufgabe 92_1 Dokumentation der Klasse *Kugel*

Auftrag init()
nachher Die Kugel ist initialisiert und besitzt schon einen Stift. Ihr Radius beträgt 20 Pixel. Sie ist allerdings noch nicht gezeichnet.

Auftrag gibFrei()
nachher Die Kugel ist verschwunden.

Auftrag bewege()
nachher Die Kugel hat sich um 0,1 Pixel bewegt.

Anfrage hPosition()
nachher Die x-Koordinate des Kugelmittelpunktes wird zurückgegeben.

Auftrag dreheBis(double pWinkel)
nachher Der zur Kugel gehörende Stift und damit die zukünftige Bewegungsrichtung des Balles hat sich entsprechend gedreht.

Auftrag bewegeBis(double px, py)
nachher Der Mittelpunkt der Kugel hat die Koordinaten (px, py).

Aufgabe 92_2

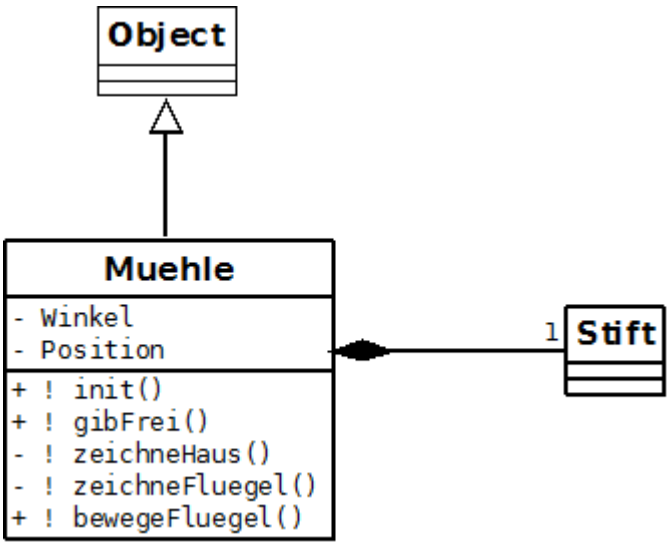
```
public void fuehreAus()
 {
  meineKugel1.bewegeBis(20,20);
  meineKugel2.bewegeBis(20,70);
  do
   {
    meineKugel1.bewege();
    if (meineKugel1.hPosition() >=
                        derBildschirm.breite() - 20 )
      meineKugel1.dreheBis(180);
```

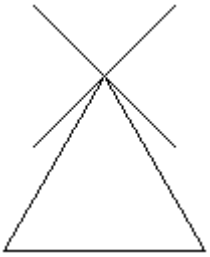
```
      if (meineKugel1.hPosition() <= 20)
        meineKugel1.dreheBis(0);
      meineKugel2.bewege();
      if (meineKugel2.hPosition() >=
                          derBildschirm.breite() - 20 )
        meineKugel2.dreheBis(180);
      if (meineKugel2.hPosition() <= 20)
        meineKugel2.dreheBis(0);
     }
   while (!meineMaus.doppelKlick());

   // Aufraeumen
   meineKugel1.gibFrei();
   meineKugel2.gibFrei();
   meineMaus.gibFrei();
   derBildschirm.gibFrei();
}
```

Aufgabe 92_3 Wie Aufgabe 2, nur der Befehl *meineKugel2.bewege* wird zweimal direkt hintereinander aufgerufen.

Aufgabe 92_4 Wie Aufgabe 2, nur mit dem Befehl *meineKugel2.bewegeBis(70, 20)*

Die Klasse *Muehle*





```
import sum.kern.*;
…….
public class Muehle extends Object
// die extends-Angabe ist überflüssig
{
     // Bezugsobjekte
     Stift hatStift;
     // Attribute
     double zWinkel, zhPosition, zvPosition;
     // Konstruktor
     public Muehle()
     {
       zhPosition = 200;
       zvPosition = 150;
       zWinkel = 45;
       hatStift = new Stift();
       zeichneFluegel();
       zeichneHaus();

     }

     // Dienste
    public void gibFrei()
     {
```

```
    hatStift.radiere();
    this.zeichneHaus();
    this.zeichneFluegel();
    hatStift.gibFrei();
   }

  private void zeichneHaus()
   {
    hatStift.hoch();
    hatStift.bewegeBis(this.zhPosition,
                                    this.zvPosition);
    hatStift.runter();
    hatStift.dreheBis(-60);
    hatStift.bewegeUm(100);
    hatStift.dreheBis(180);
    hatStift.bewegeUm(100);
    hatStift.dreheBis(60);
    hatStift.bewegeUm(100);
   }

 private void zeichneFluegel()
  {
   hatStift.hoch();
   hatStift.bewegeBis(this.zhPosition,
                                    this.zvPosition);
   hatStift.dreheBis(this.zWinkel);
   hatStift.bewegeUm(50);
   hatStift.runter();
   hatStift.bewegeUm(-100);
   hatStift.hoch();
   hatStift.bewegeBis(this.zhPosition,
                                    this.zvPosition);
   hatStift.dreheUm(90);
   hatStift.bewegeUm(50);
   hatStift.runter();
   hatStift.bewegeUm(-100);
  }
```

```
  public void bewegeFluegel()
   {
    hatStift.radiere();
    zeichneFluegel();
    zWinkel = zWinkel + 0.1;
    hatStift.normal();
    zeichneFluegel();
    zeichneHaus();
   }

}
```

Das zugehörige Hauptprogramm lautet:

```
public void fuehreAus()
 {
   do
    {
      meineMuehle.bewegeFluegel();
    }
   while (!meineMaus.doppelKlick());

  // Aufräumen
  meineMuehle.gibFrei();
  meineMaus.gibFrei();
  derBildschirm.gibFrei();
 }
```

Aufgabe

Verbessere die Klasse *Muehle* so, dass folgende Aufgaben gelöst werden können:

a) Der Konstruktor lautet nun *init(int px, int py, int pg)*. Damit lässt sich der Ort und die Größe der Mühle festlegen. Außerdem kann man nun mehrere Mühlen auf dem Bildschirm darstellen.

b) Es gibt eine Methode *setzeFarbe(int pf)*. Damit kann man die Farbe der Mühle festlegen.

c) Es gibt eine Methode *setzeGeschwindigkeit(double pv)*. Damit kann man festlegen, wie schnell sich die Mühlenflügel drehen sollen.

d) Es gibt eine Methode *setzeDrehrichtung(double pd)*. Verschiedene Mühlen können sich nun mit unterschiedlicher Drehrichtung drehen.

e) Erstelle eine Dokumentation deiner verbesserten Klasse M***uehle***!

Lösung

```
import sum.kern.*;
.....
public class Muehle extends Object
{ // die extends-Angabe ist überflüssig
 // Bezugsobjekte
 Buntstift hatBuntstift;
 // Attribute
 double zWinkel, zhPosition, zvPosition, zGroesse,
        zv, zd;
// Konstruktor
 public Muehle(int px, int py, int pg)
 {
  zhPosition = px;
  zvPosition = py;
  zGroesse = pg;
  zWinkel = 45;
  zv = 1; //Faktor für Geschwindigkeit
  zd = 1; //Drehrichtung
  hatBuntstift = new Buntstift();
 }

// Dienste
 public void gibFrei()
 {
  hatBuntstift.radiere();
  this.zeichneHaus();
  this.zeichneFluegel();
  hatBuntstift.gibFrei();
 }

 private void zeichneHaus()
 {
  hatBuntstift.hoch();
  hatBuntstift.bewegeBis(this.zhPosition,
                         this.zvPosition);
```

```
 hatBuntstift.runter();
 hatBuntstift.dreheBis(-60);
 hatBuntstift.bewegeUm(zGroesse);
 hatBuntstift.dreheBis(180);
 hatBuntstift.bewegeUm(zGroesse);
 hatBuntstift.dreheBis(60);
 hatBuntstift.bewegeUm(zGroesse);
}

private void zeichneFluegel()
{
 hatBuntstift.hoch();
 hatBuntstift.bewegeBis(this.zhPosition,
                                    this.zvPosition);
 hatBuntstift.dreheBis(this.zWinkel);
 hatBuntstift.bewegeUm(zGroesse/2);
 hatBuntstift.runter();
 hatBuntstift.bewegeUm(-zGroesse);
 hatBuntstift.hoch();
 hatBuntstift.bewegeBis(this.zhPosition,
                                    this.zvPosition);
 hatBuntstift.dreheUm(90);
 hatBuntstift.bewegeUm(zGroesse/2);
 hatBuntstift.runter();
 hatBuntstift.bewegeUm(-zGroesse);
}

public void bewegeFluegel()
{
 hatBuntstift.radiere();
 zeichneFluegel();
 zWinkel = zWinkel + 0.1*zv*zd;
 hatBuntstift.normal();
 zeichneFluegel();
 zeichneHaus();
}
```

```
 public void setzeFarbe(int pf)
 {
  hatBuntstift.setzeFarbe(pf);
  zeichneHaus();
  zeichneFluegel();
 }

 public void setzeGeschwindigkeit(double pv)
 {
  zv = pv;
 }

 public void setzeDrehrichtung(double pd)
 {
  zd = pd;
 }
}
```

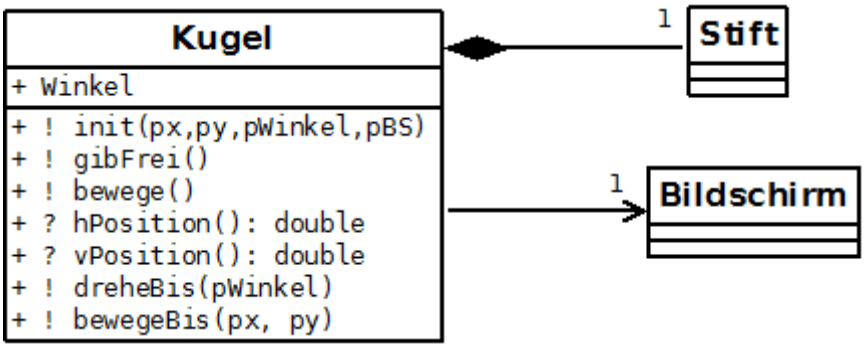
Das zugehörige Hauptprogramm lautet:

.....

```
public class Aufgabe
{
 // Objekte
 Bildschirm derBildschirm;
 Muehle meineMuehle1;
 Muehle meineMuehle2;
 Maus meineMaus;

 // Konstruktor
 public Aufgabe()
 {
  derBildschirm = new Bildschirm(400,500);
  meineMuehle1 = new Muehle(300,300,100);
  meineMuehle2 = new Muehle(150,150,50);
  meineMaus = new Maus();
 }
```

```
 // Dienste
 public void fuehreAus()
 {
  meineMuehle1.setzeFarbe(7);
  meineMuehle1.setzeGeschwindigkeit(3);
  meineMuehle2.setzeFarbe(4);
  meineMuehle2.setzeDrehrichtung(-1);
  do
    {
     meineMuehle1.bewegeFluegel();
     meineMuehle2.bewegeFluegel();
    }
  while (!meineMaus.doppelKlick());

 // Aufräumen
  meineMuehle1.gibFrei();
  meineMuehle2.gibFrei();
  meineMaus.gibFrei();
  derBildschirm.gibFrei();
 }
}
```

Die Kennt-Beziehung



Alle Kugeln bewegen sich auf **einem** Bildschirm. Wenn, wie gefordert, eine Kugel selbst erkennen soll, wann sie sich z.B. dem Bildschirmrand nähert, so muss die Kugel den Bildschirm (insbesondere dessen Ausmaße) kennen.

Weil alle Kugeln denselben Bildschirm kennen, kann dieser nicht von einer Kugel erzeugt werden, sondern jeder Kugel muss mitgeteilt werden, dass es diesen einen Bildschirm gibt, auf dem sie sich bewegen soll. Die Kugel ist also weder für die Initialisierung noch für die Freigabe des Bildschirms zuständig. **Das ist ein wesentlicher Unterschied zur hat-Beziehung.**

Die Initialisierung und Freigabe des Bildschirms wird in unserem Beispiel vom Hauptprogramm übernommen.

```
//Version mit Kennt-Beziehung
import sum.kern.*;
.........
public class Kugel
{
     // Bezugsobjekte
     private Stift hatStift;
     private Bildschirm kBildschirm;

     // Attribute
     private double zWinkel;
```

```
// Konstruktor
public Kugel(double px, double py,
                        double pWinkel, Bildschirm pBS)
 {
  hatStift = new Stift();
  kBildschirm = pBS;
  bewegeBis(px, py);
  zWinkel = pWinkel;
  hatStift.dreheBis(zWinkel);
 }

// Dienste
public void bewege()
 {
  hatStift.radiere();
  hatStift.zeichneKreis(20);
  hatStift.hoch();
  hatStift.bewegeUm(0.1);
  hatStift.normal();
  hatStift.zeichneKreis(20);
  if (this.hPosition() >= kBildschirm.breite() - 20)
    dreheBis(180-zWinkel);
  if (this.hPosition() <= 20) dreheBis(180-zWinkel);
  if (this.vPosition() >= kBildschirm.hoehe() - 20)
    dreheBis(-zWinkel);
      if (this.vPosition() <= 20) dreheBis(-zWinkel);
     }


public void bewegeBis(double px, double py)
 {
  hatStift.radiere();
  hatStift.zeichneKreis(20);
  hatStift.bewegeBis(px, py);
  hatStift.normal();
  hatStift.zeichneKreis(20);
 }
```

```
public void dreheBis(double pWinkel)
 {
  hatStift.dreheBis(pWinkel);
  zWinkel = pWinkel;
 }


public void gibFrei()
 {
  hatStift.radiere();
  hatStift.zeichneKreis(20);
  hatStift.gibFrei();
 }


public double hPosition()
 {
  return hatStift.hPosition();
 }


public double vPosition()
 {
  return hatStift.vPosition();
 }
}
```

Das zugehörige Hauptprogramm lautet nun:

```
import sum.kern.*;
……..
public class aufgabe
 {
  // Objekte
  Bildschirm derBildschirm;
  Maus meineMaus;
  Kugel meineKugel1, meineKugel2;

  // Konstruktor
  public aufgabe()
  {
   derBildschirm = new Bildschirm(800,500);
   meineMaus = new Maus();
   meineKugel1 = new Kugel(300,200,30,derBildschirm);
   meineKugel2 = new Kugel(200,200,-40,derBildschirm);
  }

  // Dienste
  public void fuehreAus()
   {
    do
     {
      meineKugel1.bewege();
      meineKugel2.bewege();
     }
    while (!meineMaus.doppelKlick());

    // Aufräumen
    meineKugel1.gibFrei();
    meineKugel2.gibFrei();
    meineMaus.gibFrei();
    derBildschirm.gibFrei();
  }
}
```

Aufgaben

107_1 Gib der Klasse *Kugel* eine neue Eigenschaft *Geschwindigkeit v*. Innerhalb der Methode *bewege* könnte man dann etwa die Anweisung *bewegeUm(v)* geben. Die Werte von v sollten ungefähr 0.1 betragen. Es sollten auch zwei neue Methoden *void setV(double pv)* und *double getV* implementiert werden.
Zwei Kugeln bewegen sich anfangs in verschiedenen Richtungen mit unterschiedlichen Geschwindigkeiten schräg über den Bildschirm. An den Bildschirmrändern werden sie entsprechend reflektiert.
Zusätzlich kann man mit der Tastatur die Kugeln kontrollieren. Bei Eingabe von „+“ oder „-“ erhöht bzw. erniedrigt sich die Geschwindigkeit der ersten Kugel. Entsprechende Tasten gibt es für die zweite Kugel.

107_2 Zwei Kugeln sollen sich über den Bildschirm bewegen und sich auch gegenseitig abstoßen können.
Gib der Klasse *Kugel* ein neues Attribut *kKugel*, sodass zwei Kugeln sich gegenseitig kennenlernen können.
Wenn die erste Kugel initialisiert wird, existiert die zweite Kugel noch nicht. Die Klasse *Kugel* benötigt also eine weitere Methode *lerneKennen(Kugel pK);*
Wenn sich zwei Kugeln zentral stoßen, so tauschen sie ihre Winkel und Geschwindigkeiten aus. Die Klasse *Kugel* benötigt also weitere Methoden *void setWinkel(double pWinkel)* und *double getWinkel*

107_3 Es soll das System *Sonne-Erde-Mond* simuliert werden.

a) In der Bildschirmmitte steht eine große Sonne. Um diese herum kreist die Erde.
   Mathematischer Hinweis: Ein im Abstand r um den Ursprung kreisender Punkt hat die Koordinaten $(r\cdot\cos(\alpha); r\cdot\sin(\alpha))$. Dabei müssen die Winkelangaben allerdings im Bogenmaß eingegeben werden
b) Um die Erde herum kreist ein noch kleinerer Mond, allerdings entsprechend schneller.
c) Zusätzlich bewegt sich die Sonne langsam horizontal über den Bildschirm.

Lösung der Aufgabe 107_2

```
import sum.kern.*;
import sum.werkzeuge.*; // wird für Wurzel und Quadrat benötigt
.........
public class Kugel
 {
  // Bezugsobjekte
  private Stift hatStift;
  private Bildschirm kBildschirm;
  private Kugel kKugel;
  private Rechner meinRechner;

  // Attribute
  private double zWinkel, zv; // zv = Geschwindigkeit

  // Konstruktor
  public Kugel(double px, double py, double pWinkel,
                                          Bildschirm pBS)
  {
   hatStift = new Stift();
   meinRechner = new Rechner();
   kBildschirm = pBS;
   bewegeBis(px, py);
   zWinkel = pWinkel;
   hatStift.dreheBis(zWinkel);
   setV(0.1);
  }

  // Dienste
  public void bewege()
   {
    hatStift.radiere();
    hatStift.zeichneKreis(20);
    hatStift.bewegeUm(zv);
    hatStift.normal();
    hatStift.zeichneKreis(20);
```

```
    if (this.hPosition() >= kBildschirm.breite() -20)
      dreheBis(180-zWinkel);
    if (this.hPosition() <= 20) dreheBis(180-zWinkel);
    if (this.vPosition() >= kBildschirm.hoehe() - 20)
      dreheBis(-zWinkel);
    if (this.vPosition() <= 20) dreheBis(-zWinkel);
    if (meinRechner.wurzel(meinRechner.quadrat(
           this.hPosition() - kKugel.hPosition()) +
           meinRechner.quadrat(this.vPosition() -
           kKugel.vPosition())) <= 40)
      {
       double hilf = this.zWinkel;
       this.setWinkel(kKugel.getWinkel());
       kKugel.setWinkel(hilf);
       hilf = this.zv;
       this.setV(kKugel.getV());
       kKugel.setV(hilf);
      }
  }


public void bewegeBis(double px, double py)
  {
   hatStift.radiere();
   hatStift.zeichneKreis(20);
   hatStift.bewegeBis(px, py);
   hatStift.normal();
   hatStift.zeichneKreis(20);
  }


public void dreheBis(double pWinkel)
  {
   hatStift.dreheBis(pWinkel);
   zWinkel = pWinkel;
  }
```

```
public void gibFrei()
 {
  hatStift.radiere();
  hatStift.zeichneKreis(20);
  hatStift.gibFrei();
 }


public double hPosition()
 {
  return hatStift.hPosition();
 }


public double vPosition()
 {
  return hatStift.vPosition();
 }


public void setV(double pv)
 {
  zv = pv;
 }


public double getV()
 {
  return zv;
 }


public void setWinkel(double pWinkel)
 {
  zWinkel = pWinkel;
  hatStift.dreheBis(pWinkel);
 }
```

```
  public double getWinkel()
   {
    return zWinkel;
   }


  public void lerneKennen(Kugel pK)
  {
   kKugel = pK;
  }
}
```

Nun folgt das Hauptprogramm:

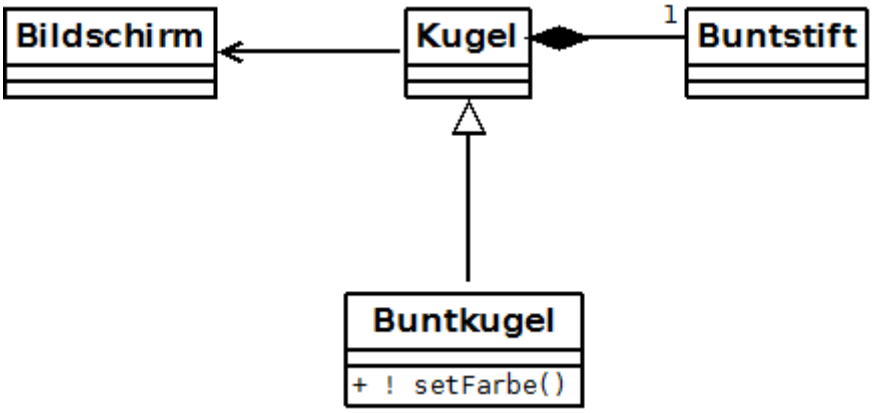
```
import sum.kern.*;
.....
public class aufgabe
 {
  // Objekte
  Bildschirm derBildschirm;
  Maus meineMaus;
  Kugel meineKugel1, meineKugel2;

  // Konstruktor
  public aufgabe()
  {
   derBildschirm = new Bildschirm(800,500);
   meineMaus = new Maus();
   meineKugel1 = new Kugel(300,200,30,derBildschirm);
   meineKugel2 = new Kugel(200,200,50,derBildschirm);
   meineKugel1.setV(0.2);
   meineKugel2.setV(0.4);
   meineKugel1.lerneKennen(meineKugel2);
   meineKugel2.lerneKennen(meineKugel1);
  }

  // Dienste
  public void fuehreAus()
   {
    do
     {
      meineKugel1.bewege();
      meineKugel2.bewege();
     }
    while (!meineMaus.doppelKlick());

  // Aufräumen
  .....
  }
}
```

Vererbung als Spezialisierung

Für das Folgende wird es wesentlich einfacher, wenn man die Klasse ***Kugel*** leicht verändert. Die zuvor ***privaten*** Attribute erhalten nun die Eigenschaft ***protected***, und der benutzte Stift soll nun ein Buntstift sein. Ansonsten ändert sich an der Klasse ***Kugel*** nichts.

```
import sum.kern.*;
.....
public class Kugel
{
    // Bezugsobjekte
    protected Buntstift hatStift;  //beachte auch Initialisierung
    protected Bildschirm kBildschirm;

    // Attribute
    protected double zWinkel;
```

Wir wollen nun Kugeln konstruieren, die sich farbig darstellen können. Dafür erweitern wir die Klasse ***Kugel*** und erhalten so eine Spezialisierung der ursprünglichen Klasse namens ***Buntkugel***. Diese Klasse ***Buntkugel*** ist eine Unterklasse der Klasse ***Kugel***.



Die Klasse ***Buntkugel*** lautet nun:

```
import sum.kern.*;
.....
public class Buntkugel extends Kugel
 {
  // keine Bezugsobjekte
  // keine Attribute
  // Konstruktor
  public Buntkugel(double px, double py,
                       double pWinkel, Bildschirm pBS)
  {
   super(px, py, pWinkel, pBS);
     // super(...) steht immer als erste Anweisung im Konstruktor einer
     // Unterklasse. Damit wird der Konstruktor der Oberklasse aufgerufen.
  }

  // Dienste
  public void setFarbe(int pFarbe)
   {
    hatStift.setzeFarbe(pFarbe);
   }
}
```

An dem zugehörigen Hauptprogramm ändert sich nur wenig:

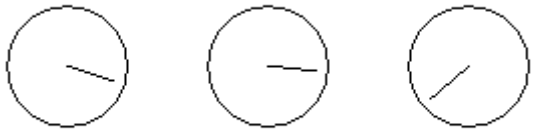
```
import sum.werkzeuge.*;  // die Klasse Farbe wird benötigt
import sum.kern.*;
.....
public class aufgabe
 {
  // Objekte
  Bildschirm derBildschirm;
  Maus meineMaus;
  Buntkugel meineKugel1, meineKugel2;
```

```
  // Konstruktor
  public aufgabe()
   {
    derBildschirm = new Bildschirm(800,500);
    meineMaus = new Maus();
    meineKugel1 = new Buntkugel(300, 200, 30,
                                          derBildschirm);
    meineKugel2 = new Buntkugel(200, 200, 50,
                                          derBildschirm);
    meineKugel1.setV(0.2);
    meineKugel2.setV(0.4);
    meineKugel1.setFarbe(Farbe.ROT);
    meineKugel2.setFarbe(Farbe.BLAU);
    meineKugel1.lerneKennen(meineKugel2);
    meineKugel2.lerneKennen(meineKugel1);
   }

  // Dienste
  public void fuehreAus()
    {
     do
      {
       meineKugel1.bewege();
       meineKugel2.bewege();
      }
     while (!meineMaus.doppelKlick());

     // Aufräumen
     meineKugel1.gibFrei();
     meineKugel2.gibFrei();
     meineMaus.gibFrei();
     derBildschirm.gibFrei();
  }
}
```

Im Folgenden werden Kugeln konstruiert, die bei ihrer Bewegung eine (leider nur kurze) Spur hinterlassen. Dazu wird für die Unterklasse *Buntkugel* die Methode ***bewege()*** neu geschrieben, welche aber die alte Methode ***bewege()*** der Oberklasse *Kugel* ausführt und nur anschließend noch den Kreismittelpunkt zeichnet:

```
public void bewege()  // Methode der Klasse Buntstift
 {
  super.bewege();
 /*  super.dienst(...) ist der Aufruf der Methode dienst der Oberklasse.
Die Anweisung super.dienst(...) kann im Gegensatz zum Aufruf des Konstruktors
der Oberklasse  super(...) an bliebiger Stelle stehen. Diese Anweisung wird
typischerweise dann benutzt, wenn ein Dienst der Oberklasse in der Unterklasse
erweitert wird.
*/
  hatStift.zeichneKreis(1);
 }
```

Übungen zur Klassenbildung



38

117_1 a) Konstruiere eine Klasse namens *AnalogeUhr*. Diese besitzt im einfachsten Fall nur einen Zeiger, welcher sich im Sekundentakt weiter bewegt. Benutze dafür die bereits früher eingeführte Klasse *Uhr* im Package *sum.werkzeuge* und insbesondere deren Methode *warte(long pDauer)*

b) Konstruiere eine Unterklasse namens *Stoppuhr*. Diese läuft nur bei gedrückter Maustaste. Das heißt, eine Stoppuhr muss die Maus kennen.

c) Konstruiere eine Unterklasse *Digitaluhr*. Diese zeigt zusätzlich noch die Zeit in digitaler Form an.

d) Konstruiere eine Unterklasse *Minutenuhr*, welche einen zweiten Zeiger besitzt, der sich genau dann einen Schritt weiterbewegt, wenn der Sekundenzeiger eine Umdrehung hinter sich gebracht hat.

e) Die Minutenuhr soll die Zeit zusätzlich auch in digitaler Form anzeigen.

117_2 Konstruiere eine Klasse namens ***Lampe***.

a) Eine Lampe – in Form eines schwarzen Kreises – kann auf dem Bildschirm gezeichnet werden. Der Radius wird bei der Erzeugung bestimmt. Sie lässt sich einschalten, dies wird durch das Ausfüllen des Kreises mit gelber Farbe gekennzeichnet, und sie lässt sich wieder ausschalten, dann wird nur der Kreisrand gezeichnet.

b) Es soll eine Unterklasse ***Farblampe*** konstruiert werden.

c) Es soll eine Unterklasse ***Bewegungsmelder*** konstruiert werden. Diese Lampe leuchtet nur dann auf, wenn ihr die Maus zu nahe kommt.

118_3 Konstruiere eine Klasse namens ***Ampel***.
Eine Ampel besteht aus einem rechteckigenGehäuse und drei farbigen Lampen. Sie besitzt eine Größe, welche schon bei der Erzeugung bestimmt wird, und eine Eigenschaft namens *phase*, welche die vier Werte rot, rot-gelb, grün und gelb annehmen kann. Außerdem gibt es eine Methode namens *weiter*, welche in die jeweils nächste Ampelphase überführt.

a) Schreibe ein Programm, welches eine Ampel darstellt, die im Sekundentakt die Phase wechselt.

b) Stelle mit dem Computer zeichnerisch eine Kreuzung mit vier Ampeln dar! Jeweils zwei gegenüberliegende Ampeln haben die gleiche Phase. Die Phasen sollen natürlich realistisch sein, d.h. wenn die Hauptstraße grün hat, muss die Ampel der Querstraße rot anzeigen.

Lösungen der Aufgaben 117_1a, b, c

Zunächst das **Hauptprogramm**:

```
import sum.werkzeuge.*; // die Klasse Uhr wird benötigt
import sum.kern.*;
.........
public class aufgabe
{
  // Objekte
  Bildschirm derBildschirm;
  Maus meineMaus;
  AnalogeUhr meineUhr;
  Stoppuhr meineStoppuhr;
  Digitaluhr meineDigitaluhr;
  Uhr zeitmesser;

  // Konstruktor
  public aufgabe()
  {
   derBildschirm = new Bildschirm(800,500);
   meineMaus = new Maus();
   meineUhr = new AnalogeUhr(100,100,0);
   meineStoppuhr = new Stoppuhr(200,100,10);
   meineStoppuhr.lerneMausKennen(meineMaus);
   meineDigitaluhr = new Digitaluhr(300,100,20);
   zeitmesser = new Uhr();
  }

  // Dienste
  public void fuehreAus()
   {
    do
     {
      meineUhr.bewege();
      meineStoppuhr.bewege();
      meineDigitaluhr.bewege();
```

```
       zeitmesser.warte(1000);
      }
      while (!meineMaus.doppelKlick());

    // Aufräumen
    meineUhr.gibFrei();
    meineStoppuhr.gibFrei();
    meineDigitaluhr.gibFrei();
    meineMaus.gibFrei();
    derBildschirm.gibFrei();
   }
}
```

Nun folgt die Klasse ***AnalogeUhr***:

```
import sum.kern.*;
.....

public class AnalogeUhr
 {
  // Bezugsobjekte
  protected Stift hatStift;
  // Attribute
  protected int zSekunde, zxM, zyM;
  // Konstruktor
  public AnalogeUhr(int px, int py, int ps)
   {
    hatStift = new Stift();
    zxM = px;
    zyM = py;
    zSekunde = ps;
    hatStift.hoch();
    hatStift.bewegeBis(zxM, zyM);
    hatStift.zeichneKreis(30);
    hatStift.runter();
   }
```

```
  // Dienste
  protected void zeichne()
   {
    hatStift.bewegeBis(zxM, zyM);
    hatStift.dreheBis(90-6*zSekunde);
    hatStift.bewegeUm(25);
   }

  public void bewege()
   {
    hatStift.radiere();
    zeichne();
    hatStift.normal();
    zeichne();
    zSekunde = zSekunde+1;
    if (zSekunde == 60) zSekunde = 0;
   }

  public void gibFrei()
   {
    hatStift.gibFrei();
   }
}
```

Nun folgt die Klasse ***Stoppuhr***:

```
import sum.kern.*;
.....

public class Stoppuhr extends AnalogeUhr
 {
  // Bezugsobjekte
  protected Maus kMaus;
  // keine Attribute

  // Konstruktor
  public Stoppuhr(int px, int py, int ps)
   {
    super(px, py, ps);
    // super(...) steht immer als erste Anweisung im Konstruktor einer
    // Unterklasse. Damit wird der Konstruktor der Oberklasse aufgerufen.
   }

  // Dienste
  public void lerneMausKennen(Maus pm)
   {
    kMaus = pm;
   }

  protected void zeichne()
   {
    if (kMaus.istGedrueckt()) super.zeichne();
   }
}
```

Nun folgt die Klasse ***Digitaluhr***:

```
import sum.kern.*;
…….
public class Digitaluhr extends AnalogeUhr
 {
  // Bezugsobjekte
  protected Buntstift hatBuntstift;
  // keine Attribute
  // Konstruktor
  public Digitaluhr(int px, int py, int ps)
   {
    super(px, py, ps);
    // super(...) steht immer als erste Anweisung im Konstruktor einer
    // Unterklasse. Damit wird der Konstruktor der Oberklasse aufgerufen.
    hatBuntstift = new Buntstift();
   }

  // Dienste
  protected void zeichne()
   {
    super.zeichne();
    hatBuntstift.bewegeBis(zxM - 35, zyM + 40);
    hatBuntstift.setzeFuellmuster(Muster.GEFUELLT);
    hatBuntstift.setzeFarbe(Farbe.WEISS);
    hatBuntstift.zeichneRechteck(60,20);
    hatBuntstift.bewegeBis(zxM - 5, zyM + 50);
    hatBuntstift.setzeFarbe(Farbe.SCHWARZ);
    if (zSekunde < 10) hatBuntstift.schreibeZahl(0);
    hatBuntstift.schreibeZahl(zSekunde);
   }

  public void gibFrei()
   {
    hatBuntstift.gibFrei();
    super.gibFrei();
    /* super.dienst(...) ist der Aufruf der Methode dienst der Oberklasse.
```

Die Anweisung super.dienst(...) kann im Gegensatz zum Aufruf des Konstruktors der Oberklasse super(...) an bliebiger Stelle stehen. Diese Anweisung wird typischerweise dann benutzt, wenn ein Dienst der Oberklasse in der Unterklasse erweitert wird.

```
*/
    }
}
```

Lösung der Aufgabe 117_2a, b, c

zunächst das Hauptprogramm:

```
import sum.kern.*;
.....
public class aufgabe
 {
  // Objekte
  Bildschirm derBildschirm;
  Maus meineMaus;
  Lampe meineLampe;
  Farblampe meineRotlampe;
  Bewegungsmelder meinBM;

  // Konstruktor
  public aufgabe()
  {
   derBildschirm = new Bildschirm(500,200);
   meineMaus = new Maus();
   meineLampe = new Lampe(100,100,20);
   meineRotlampe = new Farblampe(200,100,20);
   meineRotlampe.setzeFarbe(Farbe.ROT);
   meinBM = new Bewegungsmelder(300,100,20);
   meinBM.lerneMausKennen(meineMaus);
  }
```

```
  // Dienste
  public void fuehreAus()
   {
   // Aktionsteil
   meineLampe.machAn();
   meineRotlampe.machAn();
   do
    {
     meinBM.passAuf();
    }
   while (!meineMaus.doppelKlick());
   meineLampe.machAus();
   meineRotlampe.machAus();

   // Aufräumen
   meinBM.gibFrei();
   meineRotlampe.gibFrei();
   meineLampe.gibFrei();
   meineMaus.gibFrei();
   derBildschirm.gibFrei();
  }
}
```

nun folgt die Klasse ***Lampe***:

```
import sum.kern.*;
import sum.werkzeuge.*;
.....
public class Lampe
 {
  // Bezugsobjekte
  protected Buntstift hatBuntstift;
  // Attribute
  protected double zxM, zyM, zRadius;
  // Konstruktor
  public Lampe(double px, double py, double pR)
```

```
    {
     zxM = px;
     zyM = py;
     zRadius = pR;
     hatBuntstift = new Buntstift();
    }

  // Dienste
     protected void zeichne(int pf)
      {
       hatBuntstift.setzeFarbe(pf);
       hatBuntstift.bewegeBis(zxM, zyM);
       hatBuntstift.setzeFuellMuster(Muster.GEFUELLT);
       hatBuntstift.zeichneKreis(zRadius);
       hatBuntstift.setzeFuellMuster(
                                    Muster.DURCHSICHTIG);
       hatBuntstift.setzeFarbe(Farbe.SCHWARZ);
       hatBuntstift.zeichneKreis(zRadius);
      }

  public void machAn()
    {
     zeichne(Farbe.GELB);
    }

  public void machAus()
    {
     zeichne(Farbe.WEISS);
    }

  public void gibFrei()
    {
     hatBuntstift.gibFrei();
    }
}
```

nun folgt die Klasse ***FarbLampe***:

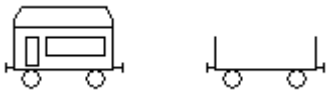
```
import sum.kern.*;
.....
public class Farblampe extends Lampe
 {
  // keine Bezugsobjekte
  // Attribute
  protected int farbe;
  // Konstruktor
  public Farblampe(double px, double py, double pR)
   {
    super(px, py, pR);
    // super(...) steht immer als erste Anweisung im Konstruktor einer
    // Unterklasse. Damit wird der Konstruktor der Oberklasse aufgerufen.
   }

  // Dienste
  public void setzeFarbe(int pf)
   {
    farbe = pf;
   }

  public void machAn()
   {
    zeichne(farbe);
   }
}
```

nun folgt die Klasse ***Bewegungsmelder***:

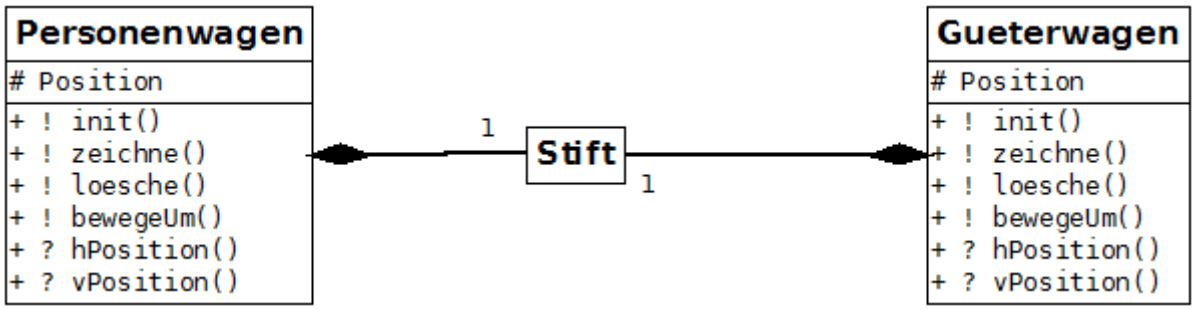
```
import sum.kern.*;
import sum.werkzeuge.*;    // die Klasse Rechner wird benötigt
……….
public class Bewegungsmelder extends Lampe
 {
  // Bezugsobjekte
  protected Maus kMaus;
  protected Rechner meinRechner;
  // keine Attribute

  // Konstruktor
  public Bewegungsmelder(double px, double py,double pR)
   {
    super(px, py, pR);
    // super(…) steht immer als erste Anweisung im Konstruktor einer
    // Unterklasse. Damit wird der Konstruktor der Oberklasse aufgerufen.
    meinRechner = new Rechner();
   }

  // Dienste
  public void lerneMausKennen(Maus pm)
   {
    kMaus = pm;
   }

  public void passAuf()
   {
    if (meinRechner.wurzel(
        meinRechner.quadrat(kMaus.hPosition() - zxM)
      + meinRechner.quadrat(kMaus.vPosition() - zyM))
        < 100)
     machAn();
    else machAus();
   }
}
```

Abstrakte Klassen als Generalisation

Wir untersuchen nun die zwei Klassen *Gueterwagen* und *Personenwagen* (letztere unterscheidet sich von der ersteren nur durch eine andere Zeichenmethode).



- Ein Wagen wird am linken Puffer ganz links in der Mitte beginnend gezeichnet.
- Die Räder haben einen Radius von 5 Pixel (= picture element)
- Der Wagen hat eine Gesamtlänge (einschließlich Puffer) von ungefähr 60 Pixel.



Dokumentation der Klasse Gueterwagen

Auftrag	**init(double phPosition, double pvPosition)**
nachher	Der Güterwagen ist initialisiert, d.h. er ist an der angegebenen Position (bestimmt durch die Koordinaten des Mittelpunktes des linken Puffers) auf dem Bildschirm dargestellt..

Auftrag	**void zeichne()**
nacher	Der Güterwagen ist an seiner aktuellen Position auf dem Bildschirm dargestellt.

Auftrag	**void loesche()**
nachher	Der Güterwagen ist nicht mehr auf dem Bildschirm dargestellt.

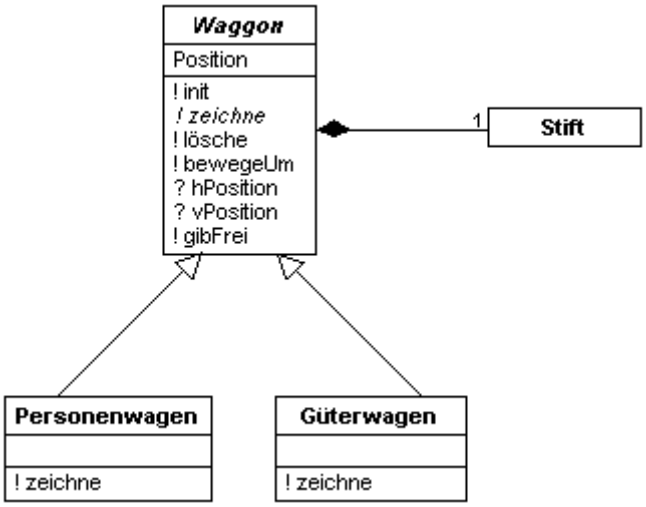
Auftrag	**void bewegeUm(double pDistanz)**
nachher	Der Güterwagen wurde um die angegebene Distanz nach links bewegt.

Anfrage	**double hPosition()**
nachher	Diese Anfrage liefert die aktuelle horizontale Position des Güterwagens.

Anfrage	**double vPosition()**
nachher	Diese Anfrage liefert die aktuelle vertikale Position des Güterwagens.

Auftrag	**void gibFrei()**
nachher	Der Güterwagen ist vom Bildschirm gelöscht und steht nicht mehr zur Verfügung.

Offensichtlich lässt sich hier aus den beiden Wagenklassen eine allgemeinere Oberklasse generalisieren:



Dokumentation der Klasse *Waggon*

Die Klasse *Waggon* ist eine sog. abstrakte Klasse. Eine abstrakte Klasse enthält mindestens eine Methode, welche in dieser Klasse nicht implementiert werden kann. Die Klasse *Waggon* dient als Oberklasse von Personenwagen und Güterwagen, die auf dem Bildschirm dargestellt und bewegt werden können. Ein Waggon befindet sich stets auf einer genau definierten Position des Bildschirms, angegeben durch die Koordinaten des Mittelpunktes des linken Puffers.

In den Unterklassen muss der Auftrag zeichne() überschrieben werden. Jeder Waggon hat einen Stift, der unter dem Namen hatStift angesprochen werden kann.

Auftrag	**init(double phPosition, double pvPosition)**
nachher	Der Waggon ist initialisiert, d.h. er ist an der angegebenen Position auf dem Bildschirm dargestellt.

Auftrag	**void *zeichne()***
nacher	Dieser Dienst ist abstrakt und muss in den Unterklassen so überschrieben werden, dass nachher der Waggon an seiner aktuellen Position auf dem Bildschirm dargestellt ist.

Auftrag	**void loesche()**
nachher	Der Waggon ist nicht mehr auf dem Bildschirm dargestellt.

Auftrag	**void bewegeUm(double pDistanz)**
nachher	Der Waggon wurde um die angegebene Distanz nach links bewegt.

Anfrage	**void double hPosition()**
nachher	Diese Anfrage liefert die aktuelle horizontale Position des Waggons.

Anfrage	**double vPosition()**
nachher	Diese Anfrage liefert die aktuelle vertikale Position des Waggons.

Auftrag	**void gibFrei()**
nachher	Der Waggon ist vom Bildschirm gelöscht und steht nicht mehr zur Verfügung.

Eine besondere Rolle spielt der Dienst ***zeichne()*** in der Klasse ***Waggon***. Weil innerhalb der Methode ***loesche()*** u.a. auch die Methode ***zeichne()*** aufgerufen wird, muss also die Klasse ***Waggon*** nicht nur die Methode ***loesche()*** sondern auch die Methode ***zeichne()*** besitzen.

Auch wenn die Methode ***zeichne()*** in der Klasse *Waggon* nicht realisiert werden kann, so muss sie doch als Vorlage für die Unterklassen vorhanden sein. Diese "nicht realisierbare" Methode heißt *abstrakt*. Sie muss in den Unterklassen *überschrieben* werden. Abstrakte Dienste werden im UML-Klassendiagramm *kursiv* gekennzeichnet.

Man sagt auch, dass die Methode *zeichne()* aufgeschoben wird, bis ihre Realisierung in einer Unterklasse erfolgen kann.

Von einer abstrakten Klasse kann es keine Objekte geben (weil nicht alle Methoden definiert sind).

Damit lassen sich offensichtlich die beiden Unterklassen wesentlich vereinfacht darstellen:

Dokumentation der Klasse Gueterwagen

Oberklasse: *Waggon*

Ein Güterwagen ist ein Waggon, der auf dem Bildschirm dargestellt und bewegt werden kann. Er befindet sich stets auf einer genau definierten Position des Bildschirms, angegeben durch die Koordinaten des Mittelpunktes des linken Puffers.

Auftrag	**void zeichne()**
nachher	Der Güterwagen ist an seiner aktuellen Position auf dem Bildschirm dargestellt.

Es folgt nun der Programmcode für die einzelnen Klassen:

```
import sum.kern.*;
……
public abstract class Waggon
 {
  // Bezugsobjekte
  protected Stift hatStift;
  // Attribute
  protected double x, y;
  // Konstruktor
  public Waggon(double px, double py)
   {
    hatStift = new Stift();
    x = px;
    y = py;
    zeichne();
   }

  // Dienste
  public abstract void zeichne();

  public void loesche()
   {
    hatStift.radiere();
    zeichne();
    hatStift.normal();
   }

  public void bewegeUm(double pDistanz)
   {
    loesche();
    x = x+pDistanz;
    zeichne();
   }

  public void gibFrei()
   {
    loesche();
```

```
    hatStift.gibFrei();
   }

  public double hPosition()
   {
    return x;
   }

  public double vPosition()
   {
    return y;
   }
}
```

nun folgt die Klasse ***Gueterwagen***:

```
import sum.kern.*;
…….
public class Gueterwagen extends Waggon
 {
  // keine Bezugsobjekte
  // keine Attribute
  // Konstruktor
  public Gueterwagen(double px, double py)
   {
    super(px, py);
   }

  // Dienste
  public void zeichne()
   {
    hatStift.hoch();    // linker Puffer
    hatStift.bewegeBis(x,y);
    hatStift.bewegeBis(x,y-2);
    hatStift.runter();
    hatStift.bewegeBis(x,y+2);
```

```
        hatStift.bewegeBis(x,y);
        hatStift.bewegeBis(x+4,y);

        hatStift.bewegeBis(x+4,y-15);     //offener Kasten
        hatStift.bewegeBis(x+4,y+1);
        hatStift.bewegeBis(x+54,y+1);
        hatStift.bewegeBis(x+54,y-15);

        hatStift.hoch();        // rechtes Rad
        hatStift.bewegeBis(x+45,y+6);
        hatStift.runter();
        hatStift.zeichneKreis(5);

        hatStift.hoch();            // linkes Rad
        hatStift.bewegeBis(x+13,y+6);
        hatStift.runter();
        hatStift.zeichneKreis(5);

        hatStift.hoch();          // rechter Puffer
        hatStift.bewegeBis(x+58,y);
        hatStift.bewegeBis(x+58,y-2);
        hatStift.runter();
        hatStift.bewegeBis(x+58,y+2);
        hatStift.bewegeBis(x+58,y);
        hatStift.bewegeBis(x+54,y);
    }

}
```

nun folgt die Klasse ***Personenwagen***:

```
import sum.kern.*;
……..
public class Personenwagen extends Waggon
 {
  // keine Bezugsobjekte
  // keine Attribute
  // Konstruktor
  public Personenwagen(double px, double py)
   {
    super(px, py);
   }

  // Dienste
  public void zeichne()
   {
    hatStift.hoch();                  // linker Puffer
    hatStift.bewegeBis(x,y);
    hatStift.bewegeBis(x,y-2);
    hatStift.runter();
    hatStift.bewegeBis(x,y+2);
    hatStift.bewegeBis(x,y);
    hatStift.bewegeBis(x+4,y);

    hatStift.bewegeBis(x+4,y-20);     // großer Kasten
    hatStift.zeichneRechteck(50,22);

    hatStift.bewegeBis(x+4,y-24);
    hatStift.bewegeBis(x+8,y-30);
    hatStift.bewegeBis(x+50,y-30);
    hatStift.bewegeBis(x+53,y-24);
    hatStift.bewegeBis(x+53,y-20);

    hatStift.hoch();                      // Tür
    hatStift.bewegeBis(x+10,y-15);
    hatStift.runter();
    hatStift.zeichneRechteck(7,15);
```

```
        hatStift.hoch();                    // Fenster
        hatStift.bewegeBis(x+20,y-15);
        hatStift.runter();
        hatStift.zeichneRechteck(30,10);

        hatStift.hoch();                    // rechtes Rad
        hatStift.bewegeBis(x+45,y+6);
        hatStift.runter();
        hatStift.zeichneKreis(5);

        hatStift.hoch();                    // linkes Rad
        hatStift.bewegeBis(x+13,y+6);
        hatStift.runter();
        hatStift.zeichneKreis(5);

        hatStift.hoch();                  // rechter Puffer
        hatStift.bewegeBis(x+58,y);
        hatStift.bewegeBis(x+58,y-2);
        hatStift.runter();
        hatStift.bewegeBis(x+58,y+2);
        hatStift.bewegeBis(x+58,y);
        hatStift.bewegeBis(x+54,y);
    }
}
```

Aufgaben

139_1 Erstelle auf dem Bildschirm einen Zug, der aus drei Waggons besteht! Der Zug soll sich von links nach rechts bewegen.

139_2 Der Zug aus der Aufgabe 1 soll an den Bildschirmrändern reflektiert werden.

139_3 Der Zug reagiert auf Tasteneingaben. Bei „+“ wird er schneller, bei „-„ langsamer.

139_4 Der Zug sollte eine zusätzliche Lokomotive als Zugmaschine erhalten. Erstelle also eine neue Unterklasse von *Waggon* namens *Lokomotive*!

139_5 Eine einzige Lokomotive soll im Kreis fahren können. Dafür müssten in der Methode *zeichne()* fast alle *bewegeBis()*-Befehle durch *bewegeUm()*-Befehle ersetzt werden. Außerdem benötigt die Lokomotive eine neue Eigenschaft *Richtung*, welche die Himmelsrichtung (in Grad) angibt, in welche die Lokomotive fährt.

139_6. Ein ganzer Zug soll im Kreis fahren können.

139_7 Erstelle eine abstrakte Oberklasse namens *Figur*, welche zwei Unterklassen namens *Quadrat* und *GleichseitigesDreieck* besitzt! Gemeinsame Eigenschaften sind der Ort des linken unteren Eckpunktes und die Länge einer Seite. Außerdem sollen unter jeder gezeichneten Figur der Umfang und die Fläche angegeben werden. Das heißt, die beiden Funktionen *Umfang()* und *Flaeche()* sind abstrakte Methoden der Oberklasse, welche in den Unterklassen konkretisiert werden müssen.

Lösung Aufgabe 139_1

```
import sum.kern.*;
…….
public class aufgabe
 {
  // Objekte
  Bildschirm derBildschirm;
  Stift meinStift;
  Personenwagen meinPersonenwagen;
  Gueterwagen meinGueterwagen;
  Maus meineMaus;

  // Konstruktor
  public aufgabe()
   {
    derBildschirm = new Bildschirm(600,300);
    meinStift = new Stift();
    meineMaus = new Maus();
    meinPersonenwagen = new Personenwagen(100,200);
    meinGueterwagen = new Gueterwagen(160,200);
   }

  // Dienste
  public void fuehreAus()
   {
    // Aktionsteil
    do
     {
      meinGueterwagen.bewegeUm(1);
      meinPersonenwagen.bewegeUm(1);
     }
    while (!meineMaus.doppelKlick());
```

```
      // Aufräumen

      meinGueterwagen.gibFrei();
      meinPersonenwagen.gibFrei();
      meinStift.gibFrei();
      meineMaus.gibFrei();
      derBildschirm.gibFrei();
    }
}
```

Klassen und Konstruktoren

Das Kennzeichen eines Konstruktors ist, dass er an einer Klasse (nicht an einer Instanz, also einem Objekt) aufgerufen wird, wodurch eine Instanz erzeugt wird (Reservierung von Speicher usw.). Der Name der Konstruktormethode ist mit dem Klassennamen identisch.
Der Aufruf der anderen, normalen Methoden ist erst möglich, wenn eine Instanz existiert

Der Konstruktor bezeichnet eine sog. Klassenmethode. Klassenmethoden sind Methoden, die nicht schon die Existenz eines Objektes dieser Klasse voraussetzen. Beim Konstruktor ist dies unmittelbar einsichtig, denn durch dessen Aufruf wird ja erst ein entsprechendes Klassenobjekt erzeugt.
Es können jedoch auch noch weitere Klassenmethoden oder sogar Klassen-konstanten existieren.

Konstruktoren müssen (selbstverständlich!) vom Typ *public* sein.

Eine Klassenmethode wird durch Voranstellen des Klassennamens aufgerufen. Beispiel: double x = Math.sqrt(2); Hierbei handelt es sich um die Klassenmethode Quadratwurzelfunktion der Klasse *Math*, welche grundsätzlich zur Verfügung steht, also nicht mit *import* eingebunden werden muss.

Obwohl die Deklaration des Konstruktors keinen Rückgabewert enthält, gibt ein Konstruktor immer einen Verweis auf das Objekt, das er erstellt, zurück. Im Konstruktor fehlt das Wort *void*.

Eine Klasse kann auch mehrere Konstruktoren haben. Diese müssen sich jedoch in der Anzahl oder im Typ der Parameter unterscheiden.

Häufig wird in einem Unterklassenkonstruktor als erstes der Konstruktor der Oberklasse aufgerufen, weil oft in der Unterklasse dieselben Initialisierungen von Attributen wie in der Oberklasse durchgeführt werden müssen. Erst danach dürfen im Unterklassenkonstruktor weitere Anweisungen folgen:

```
.....
public Unterklassenname(double a, double b)
 {
  super(a, b);    // Aufruf des Oberklassenkonstruktors
  X = 17;         // weitere Initialisierung
 }
.....
```

Im obigen Beispiel wird der Konstruktor der Oberklasse aufgerufen und die Parameter werden weitergereicht. Anschließend wird noch der Startwert für ein weiteres (wahrscheinlich Unterklassen-)Attribut festgelegt.

Falls im Konstruktor einer Unterklasse nur der Konstruktor der Oberklasse ohne Parameter (also super();) aufgerufen wird und keine weiteren Anweisungen erfolgen, so kann dieser Unterklassenkonstruktor entfallen. Er wird vom Compiler bei der Übersetzung automatisch ergänzt.

Betrachte dazu folgendes Programm:

zunächst die Implementierung der Oberklasse:

```
import sum.kern.*;
.....
public class Oberklasse
 {
  // Bezugsobjekte
  Buntstift hatBuntstift;
  // keine Attribute
  // Konstruktor
  public Oberklasse()
   {
    hatBuntstift = new Buntstift();
    hatBuntstift.bewegeBis(100,100);
    hatBuntstift.schreibeText("Diese Botschaft kommt
                                von der Oberklasse");
   }

  // für diese Demo werden keine Dienste benötigt,
 }
```

nun die Implementierung der Unterklasse:

```
// kein import;
public class Unterklasse extends Oberklasse
 {
  // keine Bezugsobjekte, Attribute und kein Konstruktor !
  // für diese Demo werden keine Dienste benötigt,
 }
```

nun folgt das Hauptprogramm:

```
import sum.kern.*;
……..
public class aufgabe
 {
  // Objekte
  Bildschirm derBildschirm;
  Maus meineMaus;
  Unterklasse UKObjekt;

  // Konstruktor
  public aufgabe()
   {
    derBildschirm = new Bildschirm(600,300);
    meineMaus = new Maus();
   }

   // Dienste
   public void fuehreAus()
    {
     // Aktionsteil
     UKObjekt = new Unterklasse();
     do
      {
       // es wird nichts getan
      }
     while (!meineMaus.doppelKlick());

     // Aufräumen
……..
}
```

Das Ergebnis dieses Programmes ist die bloße Ausgabe des Spruches
„Diese Botschaft kommt von der Oberklasse“

Enthält hingegen die Unterklasse einen eigenen Konstruktor, so wird **nur** dieser aufgerufen, der Konstruktor der Oberklasse jedoch nicht mehr, es sei denn, dass man innerhalb des Unterklassenkonstruktors mit der *super(…)*-Anweisung explizit noch einmal den Oberklassenkonstruktor aufruft.

Innerhalb einer Unterklasse kann man auch dann noch auf die Methoden der Oberklasse zugreifen, wenn diese in der Unterklasse bereits überschrieben worden sind. Dafür verwendet man die Anweisung
super.Methodenname();

Klassen und Objektzuweisungen.

Gegeben seien die beiden Klassen *Oberklasse* und *Unterklasse* sowie die beiden Deklarationen für zwei Objekte
Oberklasse OKObjekt;
Unterklasse UKObjekt;

Dann ist folgende Zuweisung zulässig:
OKObjekt = new Unterklasse();

Begründung: Aufgrund der Deklaration geht man davon aus, dass die Variable OKObjekt nur Dienste der Oberklasse nutzen wird. Dass gegebenenfalls noch weitere Methoden hinzukommen, interessiert hierbei nicht.
In diesem Fall werden also nur die syntaktisch möglichen Werteübertragungen vorgenommen, und OKObjekt greift als Instanz der Oberklasse sowieso nur auf die in der Oberklasse bereitgestellten Attribute und Dienste zu. OKObjekt kann also nicht auf die zusätzlichen Attribute oder Methoden der Unterklasse zugreifen, weil OKObjekt immer noch auf Grund seiner Deklaration eine Instanz der Oberklasse ist und bleibt.

Folgende Zuweisung ist nicht möglich:
UKObjekt = new Oberklasse();

Begründung: Aufgrund der Deklaration der Variablen UKObjekt muss man davon ausgehen, dass UKObjekt auch Dienste der Unterklasse nutzen will. Diese werden aber durch die letzte Definition nicht zur Verfügung gestellt.

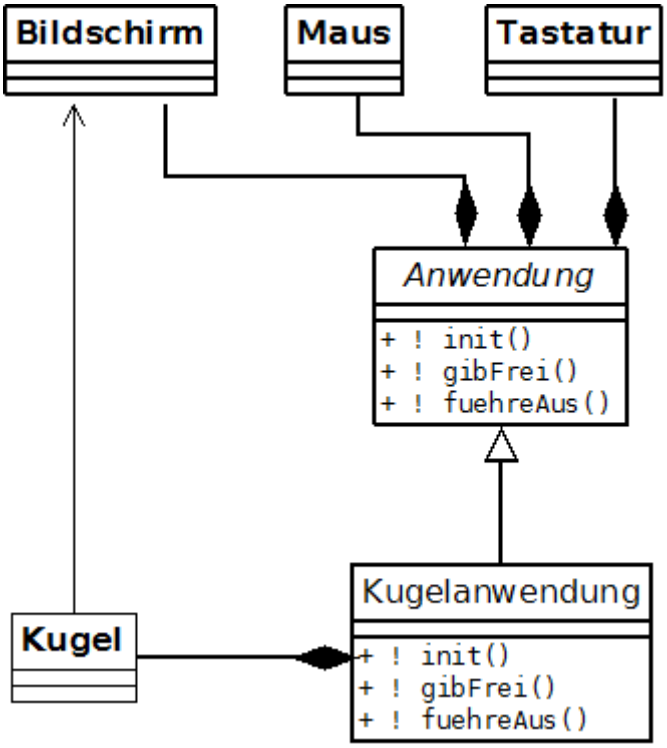
Analog:
OKObjekt = UKObjekt ist möglich; das Umgekehrte ist nicht möglich!

Die Klasse *Anwendung*

Vergleicht man alle bisherigen Hauptprogramme miteinander, so erkennt man die gemeinsame Struktur:

> Initialisierung der benötigten Objekte.
> Jeweils unterschiedlicher Ausführungsteil.
> Freigabe der benutzten Objekte.

Es ist sinnvoll, für all diese Hauptprogramme eine gemeinsame Klasse namens ***Anwendung*** zu konstruieren. Konkrete Programme werden dann als Unterklassen dieser Klasse *Anwendung* deklariert.



Dokumentation der Klasse Anwendung

Anwendung ist eine abstrakte Klasse als Basis für alle weiteren Anwendungen. Sie besitzt bereits einen Bildschirm, eine Maus, einen Stift und eine Tastatur. Bei der Realisierung von weiteren Anwendungen als Unterklasse muss insbesondere die abstrakte Methode *fuehreAus()* überschrieben werden. Außerdem muss auch der Konstruktor der Oberklasse aufgerufen werden, weil man sonst keinen Stift, Bildschirm usw. zur Verfügung hat. Auch der dienst *gibFrei()* der Oberklasse Anwendung sollte zum Schluß des Programms aufgerufen weden.
Nur in einer Unterklasse darf auf die oben aufgeführten Bezugsobjekte über die Namen „hatBildschirm“, „hatTastatur“, „hatStift“ und „hatMaus“ zugegriffen werden.

Auftrag	init()
nachher	Die Anwendung ist initialisiert. Die Objekte Bildschirm, Maus, hatStift und Tastatur sind erzeugt und initialisiert.

Auftrag	void *fuehreAus()*
	abstrakter Auftrag, der in einer Unterklasse erfüllt werden muss.

Auftrag	void *gibFrei()*
nachher	Die Anwendung steht nicht mehr zur Verfügung. Bildschirm, Maus, Stift und Tastatur wurden freigegeben.

Im Folgenden wird der Quellcode für obiges Klassen-Beziehungsdiagramm dargestellt:

Zunächst wird die Klasse Anwendung dargestellt:

```
import sum.kern.*;
public abstract class Anwendung
 {
  // Bezugsobjekte
  protected Bildschirm hatBildschirm;
  protected Stift hatStift;
  protected Maus hatMaus;
  protected Tastatur hatTastatur;
  // keine Attribute
  // Konstruktor
  public Anwendung()
   {
    hatBildschirm = new Bildschirm(600,300);
    hatMaus = new Maus();
    hatStift = new Stift();
    hatTastatur = new Tastatur();
   }

  // Dienste
  public void gibFrei()
   {
    hatTastatur.gibFrei();
    hatMaus.gibFrei();
    hatStift.gibFrei();
    hatBildschirm.gibFrei();
   }

  protected abstract void fuehreAus();
}
```

Nun folgt die Klasse Kugelanwendung. Beachte, dass natürlich die Klasse *Kugel* im Projekt vorhanden sein muss (siehe Seite 72 ff)!

```
public class Kugelanwendung extends Anwendung
 {
  // Bezugsobjekte
  Kugel meineKugel1, meineKugel2;
  // keine Attribute
  // Konstruktor
  public Kugelanwendung()
   {
    super();
    meineKugel1 = new Kugel(300,200,0,hatBildschirm);
    meineKugel2 = new Kugel(200,200,90,hatBildschirm);
   }

  // Dienste
  public void gibFrei()
   {
    meineKugel1.gibFrei();
    meineKugel2.gibFrei();
    super.gibFrei();
   }

  public void fuehreAus()
   {
    do
     {
      meineKugel1.bewege();
      meineKugel2.bewege();
     }
    while (!hatMaus.doppelKlick());
    this.gibFrei();
   }
}
```

Aufgaben

150_1 Auch für die Mühle soll eine entsprechende Mühlenanwendung geschaffen werden.

150_2 Projekt Kompass

Auf dem Bildschirm sollen ein oder mehrere Kompasse erscheinen, deren Nadeln auf die Maus als „Magnetischen Nordpol“ zeigen. Die Nadeln folgen jeder Mausbewegung.

Hinweise: Benutze die Funktion *arctan*, beachte, dass das Ergebnis im Bogenmaß angegeben wird (und nur Werte zwischen $-\pi/2$ und $+\pi/2$ besitzt), die Drehmethoden des Stiftes aber eine Gradangabe benötigen.

150_3 Projekt Autorennen

Zwei Autos sollen von einem Startpunkt zu einem Zielpunkt fahren. Die Autofahrt soll auf dem Bildschirm dargestellt werden. Verwende eine Anwendungsklasse namens ***Autofahrt***!

Start Ziel

---

o-o

---

o-o

Die Autos werden diesmal durch Zeichenketten dargestellt. Die Autos können mitteilen, ob sie am Ziel angekommen sind.

a) die Autos fahren gleich schnell.
b) es wird immer wieder zufällig bestimmt, welches Auto sich als nächstes bewegt. Auf dem Bildschirm erscheint andauernd eine Meldung wie etwa „Auto 1 liegt vorn“.
c) jedes Auto bewegt sich nur auf einen bestimmten Tastendruck hin. So können zwei Spieler ein Rennen fahren, indem sie möglichst schnell „ihre“ Taste betätigen. Um zu vermeiden, dass eine Taste permanent niedergehalten wird, bewegt sich z.B. das Auto 1 nur, wenn abwechselnd die Tasten „a“ und „s“ gedrückt werden. Zuerst bewegt es sich nur bei „a“, danach nur bei „s“, dann wieder nur bei „a“ usw. Das Auto 2 reagiert analog auf die Tasten „k“ und „l“.

d) Gib das Klassendiagramm und die Dokumentation für die Klasse ***Auto*** an!

Lösungen

Aufgabe 150_2

```
import sum.kern.*;
.....
public class Kompass
 {
  // Bezugsobjekte
  protected Stift hatStift;
  protected Maus kMaus;

  // Attribute
  double zxM, zyM;
  // Konstruktor
  public Kompass(double px, double py, Maus pm)
   {
    zxM = px;
    zyM = py;
    kMaus = pm;
    hatStift = new Stift();
    hatStift.bewegeBis(zxM,zyM);
    hatStift.zeichneKreis(20);
    hatStift.runter();
   }

  // Dienste
  protected void zeichneZeiger()
   {
    hatStift.bewegeUm(-7);
    hatStift.bewegeUm(14);
    hatStift.dreheUm(135);
    hatStift.bewegeUm(3);
    hatStift.bewegeUm(-3);
    hatStift.dreheUm(-270);
    hatStift.bewegeUm(3);
    hatStift.bewegeUm(-3);
    hatStift.dreheUm(135);
```

```
    hatStift.bewegeUm(-7);
   }

  public void ausrichten()
   {
    double winkel;
    hatStift.radiere();
    zeichneZeiger();

    if (kMaus.hPosition() - zxM != 0)
      {
       winkel = Math.atan((kMaus.vPosition() - zyM)/
              (kMaus.hPosition() - zxM)) *180/Math.PI;
       if (kMaus.vPosition() != zyM)
         if (kMaus.hPosition() > zxM) winkel = -winkel;
         else winkel = 180-winkel;
       else if (kMaus.hPosition() > zxM) winkel = 0;
            else winkel = 180;
      }
    else if (kMaus.vPosition() < zyM) winkel = 90;
         else winkel = -90;

    hatStift.dreheBis(winkel);
    hatStift.normal();
    zeichneZeiger();
   }


  public void gibFrei()
   {
    hatStift.gibFrei();
   }
}
```

nun folgt der Quelltext der Kompassanwendung:

```
.....
public class Kompassanwendung extends Anwendung
 {
  // Bezugsobjekte
  protected Kompass meinKompass1, meinKompass2;
  // keine Attribute

  // Konstruktor
  public Kompassanwendung()
   {
    super();
    meinKompass1 = new Kompass(300,200,hatMaus);
    meinKompass2 = new Kompass(200,200,hatMaus);
   }

  // Dienste
  public void fuehreAus()
   {
    do
     {
      meinKompass1.ausrichten();
      meinKompass2.ausrichten();
     }
    while (!hatMaus.doppelKlick());
    this.gibFrei();
   }


  public void gibFrei()
   {
    meinKompass1.gibFrei();
    meinKompass2.gibFrei();
    super.gibFrei();
   }
}
```

Die Klasse *EreignisanwendungNeu*

Bemerkung: in dem Modul mSuM gibt es bereits eine Klasse Ereignis-anwendung und eine Klasse Ereignisbearbeiter.

Es gibt viele Programme, die warten permanent auf ein eintretendes Ereignis wie z.B. Mausdruck, Mausloslassen, Mausklick, Mausdoppelklick oder Tastatureingabe. Ist dieses Ereignis erfolgt, so wird entsprechend reagiert. Um solche Programme zu vereinfachen, wird die Klasse *EreignisanwendungNeu* eingeführt. Ein Objekt dieser Klasse ist praktisch ein Programm, welches dauernd auf ein solches Ereignis wartet und die entsprechende Reaktion darauf von einem Unterprogramm (entspricht hier einem Objekt der Unterklasse) ausführen lässt. Diese Reaktion darf in unserem Falle nicht zu lange dauern, ansonsten könnten zwischenzeitlich eintretende Ereignisse nicht registriert werden.

```
.....
public class EreignisanwendungNeu extends Anwendung
{
 // Bezugsobjekte

 // Attribute
 private Boolean zbeendet, zMausGedrueckt;
 private double zAlteHPosition,zAlteVPosition;

 // Konstruktor
 public EreignisanwendungNeu()
  {
   super();
   zbeendet = false;
   zMausGedrueckt = false;
   zAlteHPosition = hatMaus.hPosition();
   zAlteVPosition = hatMaus.vPosition();
  }

 // Dienste
 protected void fuehreAus()
```

```
{
 do
  {
   if (hatMaus.istGedrueckt() && !zMausGedrueckt)
    {
     this.bearbeiteMausdruck(hatMaus.hPosition(),
                             hatMaus.vPosition());
     zMausGedrueckt = true;
    }
   else if (zMausGedrueckt &&
                           !hatMaus.istGedrueckt())
         {
          this.bearbeiteMausLos(hatMaus.hPosition(),
                             hatMaus.vPosition());
          zMausGedrueckt = false;
         }

   if (hatMaus.doppelKlick())
     this.bearbeiteDoppelklick(hatMaus.hPosition(),
                             hatMaus.vPosition());
   if (hatTastatur.wurdeGedrueckt())
     {
      bearbeiteTaste(hatTastatur.zeichen());
      hatTastatur.weiter();
     }

    if ((hatMaus.hPosition() != zAlteHPosition) ||
          (hatMaus.vPosition() != zAlteVPosition))
      {
       zAlteHPosition = hatMaus.hPosition();
       zAlteVPosition = hatMaus.vPosition();
       this.bearbeiteMausBewegt(zAlteHPosition,
                                zAlteVPosition);
      }

    this.bearbeiteLeerlauf();
   }
 while (! zbeendet);
}
```

```
protected void beenden()
 {
  zbeendet = true;
 }


protected void bearbeiteTaste(char pZeichen)
 {
  // soll in Unterklasse implementiert werden
 }


protected void bearbeiteMausdruck(double ph,double pv)
 {
  // soll in Unterklasse implementiert werden
 }


protected void bearbeiteMausLos(double ph, double pv)
 {
  // soll in Unterklasse implementiert werden
 }


protected void bearbeiteDoppelklick(double ph,double pv)
 {
  // soll in Unterklasse implementiert werden
 }


protected void bearbeiteMausBewegt(double ph,double pv)

 {
  // soll in Unterklasse implementiert werden
 }
```

```
  protected void bearbeiteLeerlauf()
  {
     // soll in Unterklasse implementiert werden
  }

}
```

Nun folgt der Quelltext einer Test-Ereignisanwendung:

.....

```
public class Testereignisanwendung extends
                                        EreignisanwendungNeu
 {
  // Bezugsobjekte
  // keine Attribute
  // Konstruktor
  public Testanwendung()
   {
    super();
   }

  // Dienste
  public void fuehreAus()
  {
   super.fuehreAus();
   // eigentlich unnötig, aber dann wird diese Methode in BlueJ auch direkt
   // angezeigt
  }


  protected void bearbeiteMausLos(double ph, double pv)
      {
       hatStift.bewegeBis(100,100);
       hatStift.schreibeText("Hallo");
      }


  protected void bearbeiteDoppelklick(double ph,
```

```
                                                    double pv)
    {
     beenden();
    }

}
```

Aufgabe

Implementiere eine Klasse namens Freihandzeichnen als Unterklasse der Ereignisanwendung. Wenn die Maus gedrückt und bewegt wird, soll eine entsprechende Linie gezeichnet werden. Bei Tastatureingaben wird zum Beispiel die Farbe geändert. Dafür muss die Klasse Freihandzeichnen einen eigenen Buntstift mit dem Namen (wichtig!) *hatStift* besitzen. Deshalb muss diese Klasse auch die Bibliothek *sum.kern* importieren.

Lösung:

```
import sum.kern.*;
……..
public class Freihandzeichnen extends
                                        EreignisanwendungNeu
 {
  // Bezugsobjekte
  Buntstift hatStift;
  // keine Attribute
  // Konstruktor
  public Freihandzeichnen()
   {
    super();
    hatStift = new Buntstift();
   }

// Dienste
public void fuehreAus()
```

```
 {
  super.fuehreAus();  // eigentlich unnötig, aber dann wird diese
                      //  Methode in BlueJ auch direkt angezeigt
 }

protected void bearbeiteMausdruck(double ph, double pv)
 {
  hatStift.runter();
 }

protected void bearbeiteMausLos(double ph, double pv)
 {
  hatStift.hoch();
 }

protected void bearbeiteMausBewegt(double ph,double pv)
 {
  hatStift.bewegeBis(ph, pv);
 }

protected void bearbeiteDoppelklick(double ph,double pv)
 {
  this.beenden();
 }

protected void bearbeiteTaste(char pZeichen)
 {
  switch (pZeichen)
   {
    case 'r': hatStift.setzeFarbe(Farbe.ROT); break;
    case 'b': hatStift.setzeFarbe(Farbe.BLAU); break;
    case 'g': hatStift.setzeFarbe(Farbe.GELB); break;
   }
 }

}
```

Hauptprogramm

Bis jetzt waren alle Programme nur unter der SuM-Umgebung lauffähig. Das heißt, man musste insbesondere erst ein Objekt der gewünschten Anwendungsklasse erzeugen und dann anschließend dessen Dienst *fuehreAus()* aufrufen.

Nun wird gezeigt, wie man das gewünschte Programm auch als sog. jar-Datei speichern kann, welche wie üblich durch Doppelklick gestartet wird.

Dazu benötigt man im Projekt ein Hauptprogramm. Dieses ist eine eigene Klasse, welche eigentlich nur ein Objekt der gewünschten Anwendungsklasse erzeugt und dessen *fuehreAus()*-Dienst aufruft.

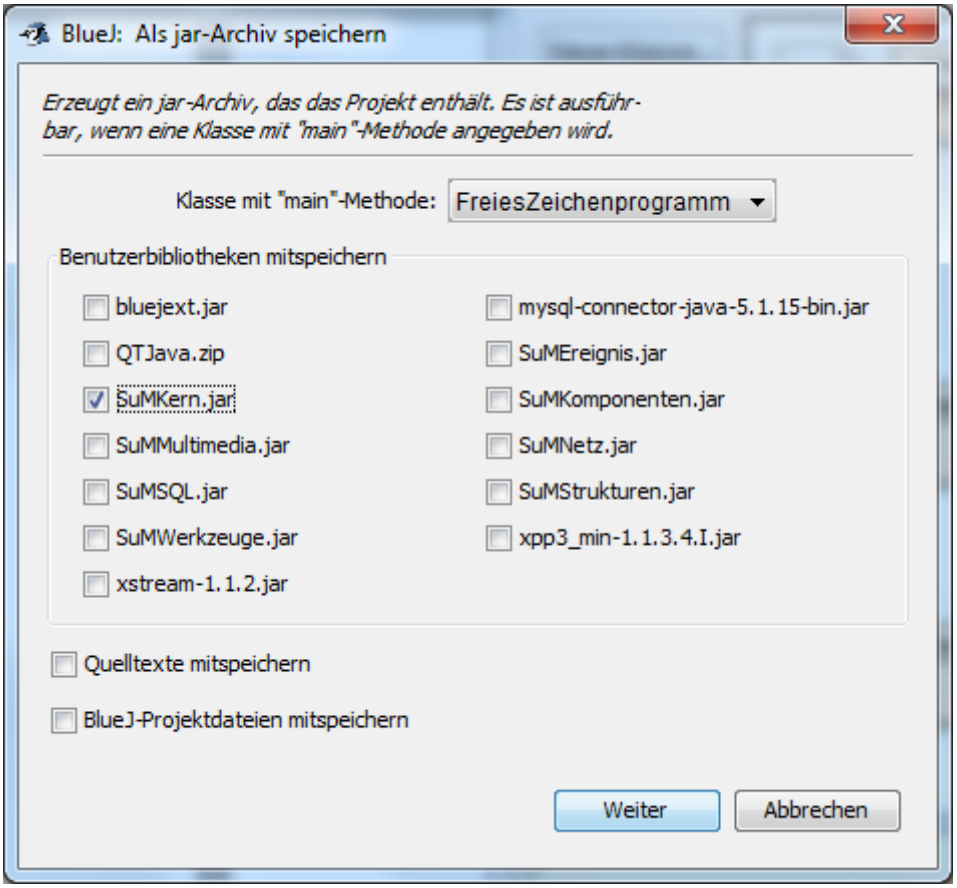
Dieses Hauptprogramm kann einen beliebigen, möglichst aussagekräftigen Namen besitzen. Sie muss jedoch einen Klassendienst haben, welcher den Namen *main* hat. Klassendienste können aufgerufen werden ohne dass ein Objekt dieser Klasse existiert. Derartige Klassendienste erkennt man an dem Wort *static*.

Das kurze Hauptprogramm bzw. diese Klasse sieht nun folgendermaßen aus (für das letzte Projekt *Freihandzeichnen*)

```
/**
 * @author
 * @version
 */
public class FreiesZeichenprogramm
{
 public static void main(String args[])
  {
   Freihandzeichnen meinProgramm =
                                  new Freihandzeichnen();
   meinProgramm.fuehreAus();
  }
}
```

Um hieraus ein eigenständiges Javaprogramm zu erzeugen, wählt man im Hauptmenü von BlueJ die Option *als jar-Archiv speichern*.

Im erscheinenden Fenster gibt man an, welche Klasse den main-Dienst enthält. Außerdem müssen natürlich alle irgendwo im Projekt benutzten Bibliotheken für das ausführbare Programm angegeben werden.



Klickt man auf *Weiter*, so kann man anschließend den Speicherort und den Namen für das Projekt angeben. Das entsprechende Unterverzeichnis enthält dann u.a. eine ausführbare *jar-Datei*.