Informatik mit Java

Eine Einführung mit **BlueJ** und der Bibliothek **Stifte und Mäuse**

Band 1

Bernard Schriek

Informatik mit Java

Eine Einführung mit

BlueJ

und der Bibliothek

Stifte und Mäuse

Band I

Nili-Verlag, Werl

Bibliografische Information der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet unter <http://dnb.ddb.de> abrufbar.

bernard.schriek@t-online.de

ISBN 3-00-017092-8

Bernard Schriek
Ostlandstr. 52
59457 Werl

Geschrieben mit dem Programm Ragtime 5.6.4 der Ragtime GmbH, Hilden
<http://www.ragtime.de>
Druck und Verarbeitung: Sächsisches Digitaldruck Zentrum GmbH, Dresden
<http://sdz-directworld.de>
Printed in Germany

Inhalt

Danksagung

Das Paket *Stifte und Mäuse (SuM)* wurde in den 90er Jahren im Rahmen der Lehrerfortbildung von Ulrich Borghoff, Dr. Jürgen Czischke, Dr. Georg Dick, Horst Hildebrecht, Dr. Ludger Humbert und Werner Ueding entwickelt. Die Bibliothek wurde zuerst in Object Pascal und Oberon geschrieben und dann auf andere objektorientierte Sprachen portiert. Die Urfassung des SuM-Kern-Pakets in Java wurde mir von Dr. Georg Dick zur Verfügung gestellt. Im Rahmen der Planung einer weiteren Fortbildungsreihe, an der ich auch beteiligt war, wurde die Bibliothek SuM-Kern um weitere Pakete erweitert. Fokke Eschen schrieb die erste Version des Programmgenerators, die dann im Rahmen einer besonderen Lernleistung von Alexander Bissaliev (Abi 2005) neu geschrieben wurde. Alle in diesem Buch behandelten Projekte wurden von den oben genannten Kollegen bei mehreren Fortbildungen zur objektorientierten Programmierung entwickelt und auch im Unterricht erprobt. Umfangreiche methodisch-didaktische Materialien wurden auf dem Bildungsserver Learnline bereitgestellt `<http://www.learnline.de/angebote/oop/>`. Dort gibt es auch ein Forum zum Gedankenaustausch. Der OOP-Bereich auf Learnline wird von Horst Hildebrecht verwaltet. Dieses Buch soll ein weiterer Baustein zum SuM-Paket sein. Ich bedanke mich bei allen Entwicklern dieses Pakets sowie den Kollegen in den verschiedenen Fortbildungsgruppen, die durch anregende Diskussionen die Weiterentwicklung unterstützt haben.

Die freie Java-Entwicklungsumgebung *BlueJ* wurde Ende der 90er Jahre an der Monash-Universität in Australien speziell für den Unterricht entwickelt und auch ständig weiter entwickelt. Besonderer Dank geht an Michael Kölling von der University of Southern Denmark, der die BlueJ-Webseiten `<http://www.bluej.org>` pflegt und mit David J. Barnes ein hervorragendes Lehrbuch zur objektorientierten Programmierung mit BlueJ geschrieben hat.

Die Programmiersprache *Java* wurde von Sun Microsystems, Santa Clara, Kalifornien entwickelt. Neben der Objektorientierung und Plattformunabhängigkeit ist die einfache, klare und konsistente Syntax und das konsequente Sprachkonzept ein besonderer Vorzug dieser Programmiersprache, die inzwischen an fast allen Hochschulen und Fachhochschulen eingesetzt wird. Mein Dank gilt allen Javaentwicklern sowie der Firma Sun, die das Java-Paket kostenlos zur Verfügung stellt.

Bedanken möchte ich mich auch bei Bastian Hafer, Philipp Hörster und Waldemar Lehmann (Jg. 11), die die Übungsaufgaben getestet haben und bei der Fehlersuche halfen.

Besonderer Dank geht an Horst Hildebrecht, der das Manuskript kritisch durchsah und mir viele Anregungen für Verbesserungen gab.

Nicht zuletzt gilt mein Dank meiner Frau, die meine stundenlangen Sitzungen am Computer mit viel Geduld ertragen hat und mich immer wieder zur Weiterarbeit ermutigte.

Werl, im September 2005

Bernard Schriek

Vorwort an Schüler

In der Informatik geht es genauso wenig um Computer, wie in der Astronomie um Teleskope.

Edsger W. Dijkstra

Sie haben sich entschlossen Informatik zu lernen. Was ist Informatik? Informatik ist eine Strukturwissenschaft, die sich mit Information und ihrer automatischen Verarbeitung beschäftigt. Das Werkzeug des Informatikers ist der Computer. Die Informatik ist eine Ingenieurwissenschaft, die sich mit mathematischen Maschinen beschäftigt. Diese Maschinen können Daten speichern, übertragen und mit der Hilfe von Algorithmen automatisch verarbeiten. Algorithmen sind eindeutige, endliche Folgen von Anweisungen zur Steuerung dieser Maschinen. Die Informatik ist inzwischen Hilfswissenschaft für fast alle anderen Fachgebiete. Die Stärke aber auch die Gefahr von Computern liegt in der Geschwindigkeit, mit der große Datenmengen verarbeitet werden können.

Damit Computer Aufgaben erledigen können, müssen sie programmiert werden. Aber so wie ein Haus zuerst von einem Architekten geplant wird, muss auch ein Computerprogramm geplant werden. Dazu gibt es verschiedene Verfahren, die Sie in diesem Buch kennen lernen werden. Natürlich benötigt man zum Hausbau auch Maurer, die ihr Geschäft verstehen. Genau so muss ein Informatiker eine oder mehrere Programmiersprachen lernen. Sie sollen mit diesem Buch die Programmiersprache Java lernen. Hinter dieser Sprache steht aber das Konzept der objektorientierten Programmierung. Dieses Konzept wird von einer Reihe von weiteren Programmiersprachen unterstützt (Object Pascal, Smalltalk, C++ u.v.m.). Wenn Sie das Konzept verstanden haben, wird es Ihnen leicht fallen, andere objektorientierte Sprachen zu lernen.

Computersprachen haben eine Syntax so, wie natürliche Sprachen eine Grammatik haben. Die Syntax dient dazu, Regeln für den Aufbau der Computersprache festzulegen. Allerdings ist die Syntax von Java im Vergleich zur Grammatik von Englisch oder Französisch um ein Vielfaches einfacher. Bevor ein Computerprogramm ausgeführt wird, muss es in die Sprache des Prozessors des konkreten Computers übersetzt werden. Dazu gibt es sogenannte Compiler, Übersetzungsprogramme, die das erstellte Programm auch auf die korrekte Syntax überprüfen. So werden schon vor der Programmausführung eventuelle Fehler gefunden. Der Compiler versucht durch Fehlermeldung dem Programmierer Hinweise über die Art des Fehlers zu geben. Am Anfang werden Sie sich oft fragen, was ist mit dieser Fehlermeldung gemeint? Oft liegt es an kleinen Schreibfehlern. Mit der Zeit werden Sie lernen, solche Fehler schnell zu finden und zu beseitigen.

Ein großes Problem bei Computerprogrammen sind allerdings logische Fehler. Sie haben bestimmt schon selbst die Erfahrung gemacht, dass ein erworbenes Softwarepaket nicht immer das tut, was es soll. Insbesondere bei kritischen Anwendungen wie Flugzeugsteuerung oder Atomkraftwerkssteuerungen, aber auch bei der Verwaltung der Auszahlungen zum Arbeitslosengeld sind solche Fehler nicht akzeptabel. Deshalb wird in der Informatikausbildung verstärkt Wert auf Verfahren gelegt, wie solche Fehler vermieden werden können. Sie werden dazu zwei Werkzeuge kennen lernen.

Welche Voraussetzungen sollte ein Schüler, eine Schülerin mitbringen, die Informatik lernen will? Neben dem Interesse am Fach sollte die Möglichkeit und die Bereitschaft

bestehen, in der Freizeit an einem Computer arbeiten. Dabei sollen möglichst alle Übungsaufgaben des Buches selbständig bearbeitet werden. Manche Aufgaben sollen auch arbeitsteilig, also von verschiedenen Personen bearbeitet werden, dabei ist aber das Hauptziel, zu lernen, klare Absprachen zu treffen, damit die getrennt entwickelten Teile zu einem funktionierenden Programm zusammengesetzt werden können. In der Softwareentwicklung arbeiten Programmierer immer in einem Team. Also ist auch Teamfähigkeit und -bereitschaft eine wichtige Voraussetzung. Computerprogramme sind die Umsetzung von Algorithmen. Algorithmen kennen Sie schon aus dem Mathematikunterricht (z.B. das Heronverfahren zur Berechnung einer Wurzel). Sie sind streng logisch aufgebaut. Spaß an der Mathematik und an logischem Denken ist also eine weitere Voraussetzung. Nicht notwendig ist Informatik- oder Computererfahrung. Dieser Kurs beginnt ganz vorn mit sehr einfachen Beispielen und Aufgaben. Trotzdem sollten Sie, auch wenn Sie schon Programmiererfahrung haben, die einfachen Aufgaben lösen und programmieren.

In diesem Band werden viele grafische Beispiele behandelt. In vereinfachter Form werden Mal- und Zeichenprogramme nachprogrammiert. Vermutlich werden Sie sich manchmal komplexere Aufgabenstellungen wünschen. Dies wird im 2. Band nachgeholt werden, denn zuerst müssen die Grundlagen gelernt werden.

Die Beispiele und Aufgaben wurden in mehreren Jahren vom Autor und mehreren anderen Lehrern im Unterricht erprobt und von mehreren Schülern vor der Drucklegung getestet. Wenn Sie aber Verbesserungsvorschläge zu machen haben oder Fehler finden, schreiben Sie eine Email an

bernard.schriek@t-online.de

Viel Erfolg bei Ihrer Informatikausbildung

Bernard Schriek

Werl, im September 2005

Vorwort an Lehrer

Dieses Buch versucht Lehrerinnen und Lehrer im Informatikunterricht zu unterstützen. Viele Lehrkräfte haben ihre erste Unterrichtserfahrung mit Pascal bzw. vor langer Zeit mit Basic gesammelt. In den 80er Jahren wurde mit großem Aufwand die strukturierte Programmierung mit Pascal unter der Informatiklehrerschaft eingeführt. Das Leitmotiv war damals: Lösen von Problemen durch Zerlegung in Teilprobleme. In den 90er Jahren gelang der objektorientierten Programmierung, die mit der Programmiersprache Smalltalk ein Nischendasein geführt hatte, mit der Verbreitung von Borland Delphi und Sun Java der Durchbruch. Insbesondere *Java* wurde zur Standardprogrammiersprache an den Universitäten und Fachhochschulen. Jetzt ist das neue Leitmotiv: Objekte schicken sich Botschaften und reagieren darauf. Die Konsequenz ist eine neue Form des Softwareentwurfs. Dazu stellt die Unified Modeling Language (UML) die passenden Diagrammformen zur Verfügung. Zur Dokumentation liefert die Java-Entwicklungsumgebung das passende Werkzeug gleich mit: JavaDoc. Die bei vielen Entwicklern unbeliebte Testphase wird durch spezielle Hilfsmittel zum automatischen Testen unterstützt. In diesem Buch wird großer Wert auf Entwurfstechniken, Dokumentation und Tests gelegt.

Lange Zeit musste der Informatikunterricht mit Werkzeugen arbeiten, die zur professionellen Softwareentwicklung gedacht waren. Inzwischen wurde mit *BlueJ* eine Java-Entwicklungsumgebung speziell für den Unterricht geschaffen. Neben einem leistungsstarken Debugger gibt es jetzt die Möglichkeit interaktiv Objekte zu erzeugen, diese zu inspizieren und ihnen Nachrichten zu schicken. BlueJ wird inzwischen an vielen Schulen und Hochschulen für die Ausbildung der Informatikstudenten genutzt.

Speziell für den Schulunterricht wurde die Bibliothek *Stifte und Mäuse* entwickelt. In ihr ist eine Turtlegrafik integriert. Die in Java komplizierte Ereignisverwaltung wurde in fertige Klassen integriert. Zur Bibliothek entstanden im Rahmen von Lehrerfortbildungen zahlreiche vielfach erprobte Unterrichtsprojekte, die in diesem Buch behandelt werden. Der vorliegende Band I behandelt die Einführung in das Klassenkonzept, Kontrollstrukturen, Vererbung, Ereignisbehandlung, Teststrategien, Dokumentation mit einer Untermege von UML und JavaDoc sowie Entwurfstechniken. Band II wird sich mit Rekursion, den verschiedenen Datenstrukturen (linearen Listen, Kellerspeichern, Bäumen, Mengen, Hashtabellen) sowie Dateiverwaltung, Sortierverfahren und der Breiten- und Tiefensuche beschäftigen. Band III handelt von der bei den Schülerinnen und Schülern sehr beliebten Netzwerkprogrammierung sowie relationalen Datenbanken (SQL). Band II erscheint im Herbst 06, Band III erscheint im Herbst 07.

An mehreren Stellen im Buch gibt es Hinweise auf Referatsthemen. Schüler können sich dazu die passenden Informationen im Internet suchen.

Zur Unterstützung der Lehrkräfte gibt es eine CD-ROM mit Musterlösungen, die gegen Nachweis der Unterrichtstätigkeit angefordert werden kann.

Die aktuellen SuM-Bibliotheken können aus dem Internet geladen werden.
<http://www.mg-werl.de/sum/>
Methodisch didaktische Erläuterungen zum Konzept von "Stiften und Mäusen" und vielen behandelten Projekten finden Sie auf dem Bildungsserver *Learnline*. Dort gibt es auch ein Forum zum Gedankenaustausch.
<http://www.learnline.de/angebote/oop/>

Kapitel 1 Installation der Bibliotheken

BlueJ ist eine Entwicklungsumgebung für Java, die an der Monash University in Australien mit dem Ziel entwickelt wurde, Schülern und Studenten den Einstieg in die objektorientierte Programmierung (OOP) zu erleichtern. BlueJ ist also eine Lernumgebung und nicht dazu gedacht, professionelle Javaprogramme zu erstellen. Dazu gibt es geeignetere Werkzeuge wie zum Beispiel Eclipse (`http://www.eclipse.org`). In den folgenden Kapiteln werden Sie die besonderen Möglichkeiten von BlueJ kennen und schätzen lernen.

Auch die SuM-Bibliothek (SuM = Stifte und Mäuse) wurde konzipiert um das Erlernen der OOP zu vereinfachen. Das SuM-Konzept wurde in den 90er Jahren von einer Gruppe von Lehrern aus Nordrhein-Westfalen entwickelt, um den Wechsel von der bis dahin üblichen imperativen Programmierung zur objektorientierten Programmierung zu unterstützen. Für verschiedene Programmiersprachen (Object-Pascal, Delphi, Java und Python) wurden die entsprechenden Bibliotheken erstellt. Im Rahmen mehrerer Lehrerfortbildungen wurden eine Reihe von Projekten erarbeitet, um die Schüler in die verschiedenen OOP-Konzepte einzuführen. Weitere ausführliche Informationen zu SuM finden Sie unter `<http://www.learnline.de/angebote/oop>`.

Um mit SuM unter BlueJ Javaprogramme entwickeln zu können, müssen zuerst drei Pakete auf ihrem Computer installiert sein:
• BlueJ, die Entwicklungsumgebung
• JDK 1.4.2 oder höher (z.B. 1.5.0), das Java Development Kit und die API-Dokumentation. Java 1.5 wird empfohlen.
• die SuM-Bibliotheken, Werkzeuge und Hilfetexte.

1.1 Installation von BlueJ

Die aktuelle Version von BlueJ können Sie sich von der BlueJ Website `<http://www.bluej.org/download/download.html>` herunterladen.

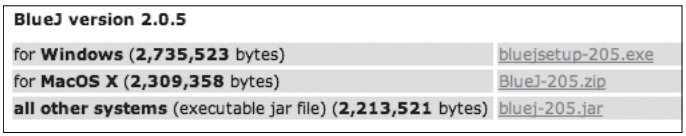


Abbildung 1.1: Download von BlueJ

Für Windows erhalten Sie einen Installer, mit dem Sie BlueJ an einen Ort ihrer Wahl installieren können. Damit allerdings später bestimmte Hilfefunktionen problemlos funktionieren, sollte BlueJ **auf der Festplatte C: im Ordner Programme** installiert werden. Benutzer mit Apple Macintosh und OSX sollten den BlueJ-Ordner **in den Ordner Programme** legen. Starten Sie BlueJ jetzt noch nicht.

1.2 Installation des JDK und der API-Dokumentation

Als Nächstes muss das aktuelle Java Development Kit (JDK) auf dem Computer installiert werden. Sie sollten aber vorher überprüfen, ob sich dieses Paket schon auf Ihrem Computer befindet. Dazu öffnen Sie bei Windows Computern die Eingabeaufforderung (das DOS-Fenster). Beim Macintosh öffnen Sie das Terminal. Geben Sie jetzt den Befehl `java -version` ein und betätigen Sie die Returntaste. Falls das JDK installiert ist, erhalten Sie eine Meldung über die installierte Javaversion. Sie sollte mindestens die Versionsnummer 1.4.2 besitzen. Falls Sie allerdings eine Fehlermeldung erhalten, müssen Sie das JDK erst noch auf ihrem Rechner installieren. Windowsbenutzer können sich das JDK von <http://java.sun.com/j2se/1.4.2/download.htm> bzw. <http://java.sun.com/j2se/1.5.0/download.jsp> herunterladen (J2SE = Java 2 Standard Edition) und installieren. Beachten Sie, dass Sie `JSE 1.4.2` oder höher auswählen (und nicht J2EE = Java 2 Enterprise Edition). Macintoshbenutzer können sich das aktuelle JDK von <http://www.apple.com/java/> herunterladen. Normalerweise ist aber das JDK schon unter OSX vorinstalliert.

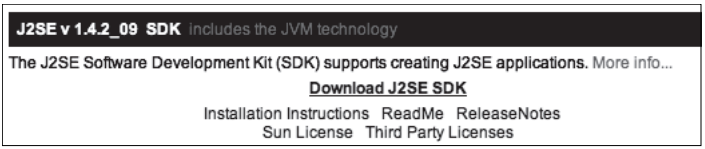


Abbildung 1.2: Download von Java (J2SE SDK)

Für beide Plattformen (Mac und Windows) benötigen Sie jetzt noch die Dokumentation der Javastandardbibliotheken, also die API-Dokumentation (API = Application Programming Interface). Diese finden Sie im gleichen Fenster, von dem aus Sie das JDK geladen haben, etwas weiter unten
<http://java.sun.com/j2se/1.4.2/download.htm#docs> bzw.
<http://java.sun.com/j2se/1.5.0/download.jsp#docs>.

Abbildung 1.3: Download der Dokumentation zu Java

Nach dem Download erhalten Sie einen Ordner, dem Sie den Namen `docs` geben und den Sie in den BlueJ-Ordner legen.

1.3 Installation der SuM-Bibliotheken

Die aktuellen SuM-Bibliotheken finden Sie unter <http://www.mg-werl.de/sum/>. Wenn Sie die zip-Datei runter geladen und entpackt haben, erhalten Sie einen Ordner `SuMWin 6.x` bzw. `SuMMac 6.x`. Dessen Inhalt muss an bestimmte Stellen kopiert werden:

- Den Ordner `doc` legen Sie in den BlueJ-Ordner.
- Den Ordner `docs` legen Sie in den BlueJ-Ordner.

Windows-Benutzer öffnen jetzt den Ordner `lib` im BlueJ-Ordner.

Mac-Benutzer machen einen Rechtsklick bzw. ctrl-Klick auf das Programm BlueJ im BlueJ-Ordner und wählen im Kontextmenü `Paketinhalt zeigen`. Doppelklicken Sie `Contents - Resources - Java`.

- Ersetzen Sie den Ordner `german` durch den Ordner mit dem gleichen Namen.
- Ersetzen die Datei `bluej.defs` durch die Datei mit dem gleichen Namen.
- Legen Sie `SuMGenerator.jar` in den Ordner `extensions`.
- Legen Sie die folgenden acht jar-Dateien in den Ordner `userlib`: `SuMKern.jar`, `SuMEreignis.jar`, `SuMWerkzeuge.jar`, `SuMKomponenten.jar`, `SuMNetz.jar`, `SuMSql.jar`, `SuMStrukturen.jar`, `SuMMultimedia.jar`.

Schließen Sie die Ordner und starten Sie BlueJ. Falls Sie mehrere JDKs auf Ihrem Rechner installiert haben, werden Sie gefragt, mit welchem JDK Sie arbeiten wollen.

1.4 Test der Installation

Als Nächstes sollen Sie kontrollieren, ob alle Komponenten korrekt installiert wurden.

Öffnen Sie die BlueJ-Einstellungen und wählen Sie `Bibliotheken`. Im unteren Teil des Fensters sollte dann stehen, dass die acht SuM-Bibliotheken geladen wurden. Schließen Sie die Einstellungen und wählen Sie im Hilfemenü `Dokumentation sum.kern`. Jetzt sollte sich der Internet-Browser öffnen und die Dokumentation zum Paket `sum.kern` anzeigen. Schließen Sie das Browserfenster. Falls die Dokumentationen nicht angezeigt werden, haben Sie den BlueJ-Ordner nicht an die oben angegebenen Stelle gelegt oder der Ordner `doc` ist nicht im BlueJ-Ordner.

Wählen Sie im Hilfemenü `Java Klassenbibliotheken`. Im Internetbrowser sollte jetzt die Javadokumentation aus dem Ordner `docs/api/index.html`, der im BlueJ-Ordner liegt, angezeigt werden. Falls dies nicht der Fall ist, können Sie sich im Einstellungsfenster von BlueJ unter `Diverses` bei der Eigabe `URL der Javadokumentation` zur oben stehenden Seite durchklicken. Schließen Sie das Browserfenster.

Wählen Sie im Menü `Werkzeuge` den `SuM-Programmgenerator`. Jetzt sollte sich ein Fenster öffnen, in dem Sie verschiedene Komponenten anlegen und das zugehörige SuM-Programm erzeugen können.

Stellen Sie unter `Einstellungen - Editor - Zeilennummern anzeigen` an.

Damit ist die Installation beendet.

Kapitel 2 Klassen und Objekte

In diesem Kapitel sollen Sie lernen:

- was eine Klasse ist
- was ein Objekt ist
- wie man ein Objekt erzeugt
- wie man die Dienste eines Objekts aufruft
- Anfragen und Aufträge zu unterscheiden
- wie man die Dokumentation einer Klasse aufruft
- wie man die Direkteingabe für Dienstaufrufe benutzt

Dieses Kapitel erläutert die beiden wichtigsten Begriffe der objektorientierten Programmierung(OOP): Klassen und Objekte. Diese Begriffe werden in allen weiteren Kapiteln ständig auftauchen und ihr Verständnis ist unabdingbar, um die weiteren Abschnitte dieses Buches erfolgreich bearbeiten zu können.

2.1 Klassen und Objekte

Wenn ein Architekt ein Haus entwirft, so erstellt er eine Zeichnung, einen Plan des Hauses. Dieser Plan ist dann die Grundlage für die Errichtung des Hauses. Es ist auch möglich zu einem Plan mehrere Häuser zu bauen, die dann natürlich gleich aussehen. Man hat es hier mit einer Vorlage, dem Plan, und konkreten Realisierungen, den Häusern, zu tun. In der objektorientierten Programmierung entspricht dem Plan eine Klasse und den konkreten Häusern entsprechen Objekte. Man kann auch sagen: Es gibt mehrere Exemplare des Hauses.

Ein Computerprogramm in einer objektorientierten Sprache versucht in der Regel einen Ausschnitt der realen Welt im Computer abzubilden. Dieser Ausschnitt besteht normalerweise aus mehreren Objekten, die miteinander kommunizieren bzw. Nachrichten austauschen. Viele dieser Objekte gehören einer gemeinsamen Gruppe an, die man Klasse nennt. Jedes Objekt gehört zu einer bestimmten Klasse. Mehrere Objekte können einer gemeinsamen Klasse angehören. Man kann eine **Klasse als Bauplan für Objekte** auffassen.

Wenn man eine Computeranlage betrachtet, fallen einem sofort mehrere Objekte auf: der Bildschirm, die Maus, die Tastatur. Wenn auf dem Bildschirm etwas gezeichnet werden soll, so stellt man sich in Gedanken einen Stift vor, mit dem man auf dem Bildschirm zeichnen kann. Eigentlich kann man ja sogar mit mehreren Stiften gleichzeitig auf dem Bildschirm zeichnen. Diese vier Klassen `Bildschirm`, `Stift`, `Maus` und `Tastatur` werden Sie in den nächsten Abschnitten etwas genauer untersuchen. BlueJ stellt Ihnen dazu die geeigneten Werkzeuge zur Verfügung.

2.2 Objekte erzeugen

Starten Sie BlueJ und erzeugen Sie ein neues Projekt mit dem Namen `Eins`. Jetzt soll ein neues Objekt der Klasse Bildschirm erzeugt werden. Die Klasse Bildschirm ist Teil der SuM-Bibliothek. Deshalb rufen Sie im Menü `Werkzeuge - Klasse aus Bibliothek verwenden` auf. Geben Sie `sum.kern.Bildschirm` ein und drücken Sie die Returntaste.

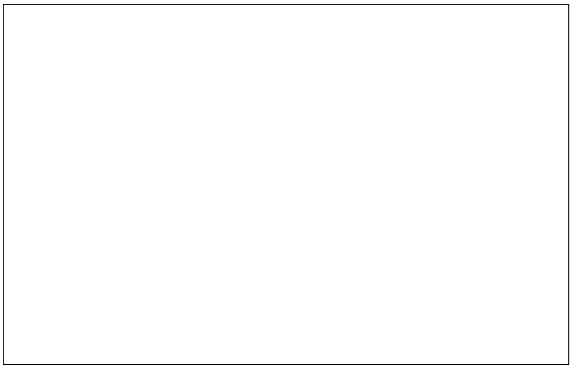


Abbildung 2.1:
Auswahl des Konstruktors

Sie haben jetzt drei Möglichkeiten, ein neues Objekt der Klasse Bildschirm zu erzeugen: `sum.kern.Bildschirm(int, int, int, int)` ermöglicht es, mit vier ganzen Zahlen (int = ganze Zahl) die linke obere Ecke, die Breite und die Höhe des Bildschirms anzugeben. `sum.kern.Bildschirm(int, int)` ermöglicht es, mit zwei ganzen Zahlen die Breite und die Höhe des Bildschirms anzugeben. `sum.kern.Bildschirm()` erzeugt einen möglichst großen Bildschirm. Wählen Sie die erste Möglichkeit mit vier ganzen Zahlen. Als Nächstes müssen Sie jetzt den Namen für das Objekt, die Koordinaten für die linke, obere Ecke, die Breite und die Höhe des Bildschirms angeben.

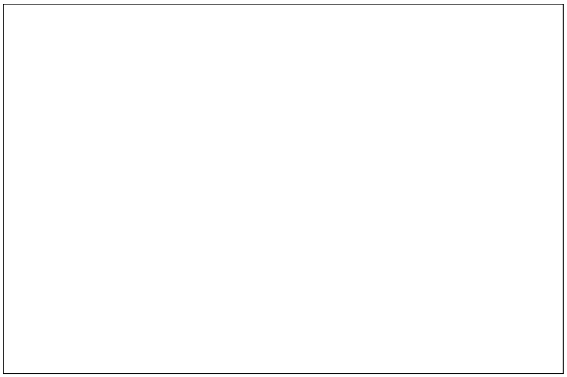


Abbildung 2.2:
Eingabe des Objektbezeichners und der Parameter

Klicken Sie in den Ok-Knopf und das Objekt `bildschirm1` der Klasse `Bildschirm` wird erzeugt. Manche Programmierer nennen das Objekt auch *Instanz* der Klasse und das Erzeugen eines Objekts infolgedessen *Instanziieren*. Da dieser Begriff auf Deutsch

meistens bei den Juristen benutzt wird, sollten Sie allerdings besser die Bezeichnung *Exemplar* verwenden.

Sie sehen ein neues leeres Fenster mit dem Titel `SuM-Programm`. Im BlueJ-Projektfenster sehen Sie unten links ein rotes Symbol für das neue Objekt, ein Exemplar der Klasse Bildschirm.

Abbildung 2.3:
Objekt bildschirm1
der Klasse Bildschirm

Der untere Bereich des Projektfensters wird als *Werkbank* (engl. work bench) bezeichnet. In ihm werden Objekte angezeigt, im oberen Bereich werden Klassen angezeigt.

2.3 Dienste aufrufen

Klicken Sie mit der rechten Maustaste (Macintosh ctrl-Klick) in das Objektsymbol und ein Kontextmenü klappt auf, das die Dienste anzeigt, die das Objekt `bildschirm1` zur Verfügung stellt.

Abbildung 2.4:
Dienste des
Objekts bildschirm1

Wählen Sie `int breite()` und die Breite des Bildschirms (Fensters) wird angezeigt.

Abbildung 2.5:
Rückgabewert der
Anfrage breite() an
den Bildschirm

Der Klassenname `Bildschirm` als Bezeichner für ein Fenster ist historisch zu erklären. Als das Paket *Stifte und Mäuse* entwickelt wurde, war der Bildschirm der gesamte sichtbare Bildschirm. Inzwischen sind die modernen Betriebssysteme allerdings fensterorien-

tiert, der Name `Bildschirm` wurde aber beibehalten.
Beachten Sie: Klassenbezeichner beginnen mit einem Großbuchstaben z.B. `Bildschirm`, Objektbezeichner beginnen mit einem Kleinbuchstaben z.B. `bildschirm1`. Dies ist eine Vereinbarung, an die sich die meisten Java-Programmierer auf der Welt halten.

Übung 2.1 Rufen Sie den Dienst `void setzeFarbe(int)` des Objekts `bildschirm1` auf und geben Sie als Farbwert die Zahl 10 an. Was ändert sich im Fenster SuM-Programm?

Übung 2.2 Finden Sie durch Veränderung der Bildschirmfarbe heraus, wie viele Farben vordefiniert sind und welche Zahlen den verschiedenen Farben zugeordnet sind. Beginnen Sie mit der Zahl 0.

Übung 2.3 Verändern Sie die Größe des Fensters SuM-Programm durch Ziehen mit der Maus. Prüfen Sie, ob sich die Breite und Höhe des Objekts `bildschirm1` geändert hat.

Nicht alle Dienste der Klasse `Bildschirm` eignen sich zum direkten Aufruf. Insbesondere sollten Sie den Dienst `gibFrei` nicht aufrufen. Seine Bedeutung wird in den nächsten Kapiteln erklärt.

2.4 Klassendokumentationen ansehen

Jetzt sollen Sie als zweite Klasse den Stift etwas näher kennenlernen. Ein Stift dient dazu, auf dem Bildschirm zu malen. Voraussetzung für die Erzeugung eines Stifts ist die Existenz eines Bildschirms. Erzeugen Sie also vorher ein Bildschirmobjekt mit dem Namen `bildschirm1`.

Ein Stift wird erzeugt, indem Sie im Menü `Werkzeuge - Klasse aus Bibliothek verwenden` aufrufen, `sum.kern.Stift` eingeben und Return drücken. Dann können Sie mit `sum.kern.Stift()` einen neues Objekt der Klasse `Stift` mit dem Namen `stift1` erzeugen.

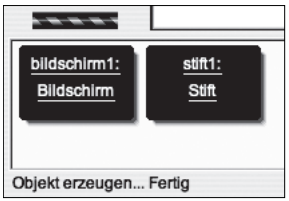
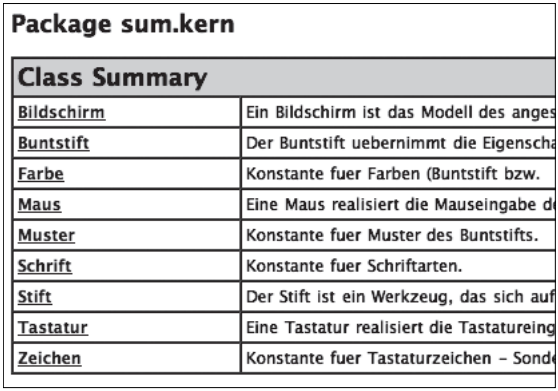


Abbildung 2.6:
Objekt stift1
der Klasse Stift

Wenn man mit einem Stift zeichnen will, muss man ihn vorher auf das Papier (den Bildschirm) absenken. Dazu dient der Dienst `void runter()`. Wenn man den Stift jetzt bewegt, wird auf dem Bildschirm eine schwarze Linie gezogen. Der Dienst `void bewegeBis(double, double)` dient dazu, den Stift auf einen bestimmten Punkt, dessen beide Koordinaten angegeben werden müssen, zu bewegen. Das Wort `double` bedeutet, dass die Koordinaten auch Dezimalzahlen (Kommazahlen) sein dürfen. Beachten Sie, dass Dezimalzahlen in Java mit einem Punkt (statt Komma) geschrieben werden. Ein

Stift hat auch eine Richtung. Er kann um einen bestimmten Winkel gedreht werden, er kann auf einen bestimmten Winkel gedreht werden. Wenn der Stift mit `void hoch()` angehoben wurde, zeichnet er beim Bewegen nicht mehr. Er muss erst wieder mit `void runter()` gesenkt werden. Beim Aufruf der Dienste erkennen Sie sofort, dass ein Stift viel mehr Dienste zur Verfügung stellt als ein Bildschirm. Um zu verstehen, was der Aufruf dieser Dienste bewirkt, sollen Sie jetzt die Klassendokumentation der beiden Klassen `Bildschirm` und `Stift` ansehen. Diese Klassendokumentation wirkt auf den ersten Blick etwas verwirrend. Im Laufe der Arbeit mit Java werden Sie sich aber schnell an diese Form der Darstellung gewöhnen.

Rufen Sie im Menü `Hilfe` die Zeile `Dokumentation sum.kern` auf. Der Internetbrowser wird gestartet und Sie erhalten eine Seite mit sämtlichen Klassen des SuMKern-Pakets.

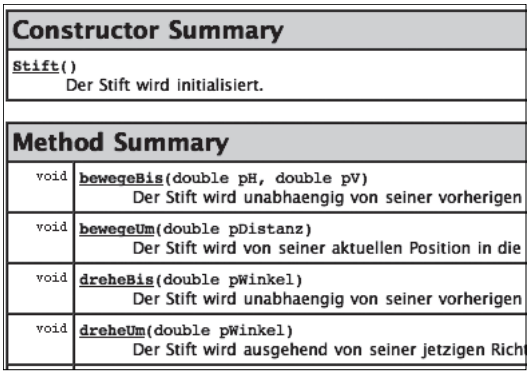


Package sum.kern

Class Summary	
Bildschirm	Ein Bildschirm ist das Modell des anges
Buntstift	Der Buntstift uebernimmt die Eigenscha
Farbe	Konstante fuer Farben (Buntstift bzw.
Maus	Eine Maus realisiert die Mauseingabe d
Muster	Konstante fuer Muster des Buntstifts.
Schrift	Konstante fuer Schriftarten.
Stift	Der Stift ist ein Werkzeug, das sich auf
Tastatur	Eine Tastatur realisiert die Tastatureing
Zeichen	Konstante fuer Tastaturzeichen – Sond

Abbildung 2.7: Übersicht der SuMKern-Klassen

Wenn Sie sich dann die Klasse `Stift` ansehen, erhalten Sie eine Auflistung sämtlicher Dienste und der dazugehörenden Erläuterungen.



Constructor Summary

`Stift()`
Der Stift wird initialisiert.

Method Summary

void	`bewegeBis(double pH, double pV)` Der Stift wird unabhaengig von seiner vorherigen
void	`bewegeUm(double pDistanz)` Der Stift wird von seiner aktuellen Position in die
void	`dreheBis(double pWinkel)` Der Stift wird unabhaengig von seiner vorherigen
void	`dreheUm(double pWinkel)` Der Stift wird ausgehend von seiner jetzigen Richt

Abbildung 2.8: Ein Teil der Dienste der Klasse Stift

Diese Dienste werden oft auch *Methoden* genannt. Wir werden aber den Begriff *Dienst* benutzen.

Der Konstruktor ist ein spezieller Dienst, der zum Erzeugen eines Objekts genutzt wird. Manche Klassen, z.B. die Klasse `Bildschirm`, haben mehrere Konstruktoren, die sich in der Anzahl und Art der Parameter (Informationen in der Klammer) unterscheiden.

2.5 Anfragen und Aufträge

Es gibt grundsätzlich zwei Arten von Diensten: Anfragen und Aufträge.
Anfragen liefern eine Antwort zurück. Zum Beispiel beantwortet der Bildschirm die Anfrage `int hoehe()` mit einer ganzen Zahl (`int`), die die Höhe des Bildschirms (Fensters) angibt. Ein Stift antwortet auf die Anfrage `double hPosition()` mit einer Dezimalzahl (`double`), die der aktuellen horizontalen Position des Stifts entspricht. Das erste Wort (`int` bzw. `double`) der Anfrage nennt man den *Ergebnistyp* der Anfrage.

Ein Auftrag liefert keine Antwort. Schickt man einem Objekt einen Auftrage, so erledigt das Objekt den Auftrag. Der Stift bewegt sich, der Bildschirm ändert seine Farbe. Das Wort `void` bedeutet, dass kein Ergebnis zurückgegeben wird, der Dienst ist dann ein Auftrag.

Anfragen werden auch als *sondierende Methoden* und Aufträge als *verändernde Methoden* bezeichnet.

Dienste, also Anfragen und Aufträge, haben Bezeichner, die mit einem Kleinbuchstaben beginnen. Falls der Bezeichner aus mehreren Worten besteht, werden diese Worte zusammen geschrieben, die weiteren Worte beginnen aber jeweils mit einem Großbuchstaben z. B. `void loescheAlles()`. Diese Art des Schreibens von zusammengesetzten Bezeichnern nennt man *CamelCase*. Bezeichner von Diensten enden immer mit einer Klammer.

Übung 2.4 Informieren Sie sich im Internet über die Herkunft und Bedeutung des Begriffs *CamelCase*.

Übung 2.5 Beenden Sie das Programm BlueJ und starten Sie es anschließend erneut. Erzeugen Sie einen Bildschirm und einen Stift.

Übung 2.6 Zeichnen Sie mit dem Stift auf dem Bildschirm ein gleichseitiges Dreieck mit der Seitenlänge 100. Falls die Zeichnung misslingt, können Sie dem Bildschirm den Auftrag `loescheAlles()` schicken und anschließend neu zeichnen.

Übung 2.7 Zeichnen Sie ein Quadrat der Seitenlänge 200 und zeichnen Sie die beiden Diagonalen ein.

2.6 Die Punktnotation für Dienstaufrufe

Es gibt in BlueJ noch eine weitere Möglichkeit Anfragen oder Aufträge an Objekte zu senden. Dazu sollten Sie vorher wie oben erläutert, einen Bildschirm und einen Stift erzeugt haben. Im Menü `Ansicht` wählen Sie `Direkteingabe anzeigen`. Der untere Bereich des Projektfensters, die Werkbank, wird jetzt in zwei Hälften geteilt. Schreiben Sie in die rechte Hälfte `bildschirm1.setzeFarbe(7)` und drücken Sie die Returntaste. Fügen Sie weitere Dienstaufrufe hinzu wie in Abbildung 2.9 angezeigt. Beobachten

Sie die Auswirkungen im Fenster SuM-Programm.

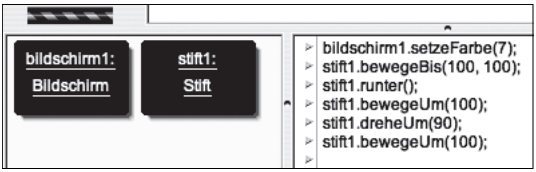


Abbildung 2.9:
Direktaufruf von Diensten

Es ist auch möglich, mehrere Anweisungen hintereinander zu schreiben, sie müssen dann aber durch ein Semikolon getrennt werden.

Die Syntax der Anweisungen ist einfach: Zuerst wird das Objekt angegeben, dem die Nachricht geschickt werden soll, dann folgt, durch einen Punkt getrennt, die Bezeichnung der Nachricht mit den zugehörigen Parametern in Klammern. Die Klammern sind zwingend notwendig, auch wenn keine Parameter erwartet werden, z.B. `stift1.runter()`. Diese Schreibweise nennt man *Punktnotation*.

2.7 Zusammenfassung

In diesem Kapitel wurden viele neue Begriffe eingeführt. Diese Begriffe werden Sie in den nächsten Kapiteln immer wieder vorfinden und so wird sich ihre Bedeutung immer stärker einprägen.

Sie haben gelernt, wie man mit BlueJ Objekte aus Klassen erzeugt, die eine Bibliothek, hier der Bibliothek `sum.kern`, zur Verfügung stellt. Zum JDK gehört eine umfangreiche Klassenbibliothek, die Tausende von Klassen enthält. Diese Klassen sind zu Paketen (packages) zusammengefasst. Die SuM-Bibliothek wurde speziell für den Schulbereich entwickelt, um einfach zu handhabende Objekte zu erhalten. In diesem Buch werden Sie das Paket `sum.kern` näher kennen lernen, das die Basisklassen `Bildschirm`, `Stift`, `Maus` usw. enthält. Sie haben gelernt, dass Klassen Dienste zur Verfügung stellen, so dass man Objekten Botschaften schicken kann, die diese Dienste aufrufen.

Neue Begriffe in diesem Kapitel

• **Klasse** Eine Klasse ist der Bauplan für ein oder mehrere Objekte. In einer Klasse werden Dienste zur Verfügung gestellt. Klassennamen beginnen mit einem Großbuchstaben.

• **Objekt** Ein Objekt ist die Realisierung einer Klasse. Jedes Objekt gehört zu einer Klasse. Die Dienste eines Objekts können aufgerufen werden. Objektnamen beginnen mit Kleinbuchstaben.

• **Exemplar** Ein Exemplar einer Klasse ist eine Realisierung dieser Klasse, also ein Objekt.

• **Dienst** Dienste beschreiben Fähigkeiten von Objekten. Dienste werden in Klassen definiert. Es gibt zwei Arten von Diensten: Anfragen und Aufträge. Dienstbezeichner beginnen mit einem Kleinbuchstaben und enden mit einer evtl. leeren Klammer für Parameter.

• **Anfrage** Eine Anfrage ist ein Dienst, den eine Klasse für ihre Objekte zur Verfügung stellt. Eine Anfrage liefert ein Ergebnis zurück.

• **Auftrag** Ein Auftrag ist ein Dienst, den eine Klasse für ihre Objekte zur Verfügung stellt. Wenn ein Objekt einen Auftrag erhält, tut es etwas, es liefert kein Ergebnis zurück.

• **Konstruktor** Der Konstruktor ist ein spezieller Dienst einer Klasse, der bei der Erzeugung eines Objekts dieser Klasse ausgeführt wird. Er wird mit dem Schlüsselwort `new` aufgerufen. Viele Klassen haben mehrere Konstruktoren, die sich in der Anzahl oder der Art ihrer Parameter unterscheiden.

• **Werkbank** (engl. work bench), unterer Bereich des Projektfensters. Dort werden mit `new` erzeugte Objekte dargestellt. Diese Objekte kann man dann inspizieren und ihnen direkt Anfragen und Aufträge schicken.

• **Punktnotation** Wenn der Dienst eines Objekts aufgerufen wird, sagt man auch, dass dem Objekt eine Botschaft geschickt wird. Dazu benutzt man die Schreibweise `Objekt.Dienst(evtl. Parameter)`, die als Punktnotation bezeichnet wird.

• **Parameter** Parameter ermöglichen es, Dienste flexibel zu gestalten. Parameter werden in Klammern hinter den Dienst geschrieben und beeinflussen die Art, wie der Dienst ausgeführt wird.

• **CamelCase** Schreibweise für Bezeichner in Java, die aus mehreren Worten zusammengesetzt sind.

Java-Bezeichner

• **int** ganze Zahl z.B.: `-123`

• **double** Dezimalzahl (Kommazahl) z.B. `3.1415`

• **void** direkt übersetzt: `Lücke, nichts`. Gibt an, dass ein Dienst keinen Wert zurück liefert, also ein Auftrag ist.

• **new** Aufruf des Konstruktors einer Klasse. Dient zum Erzeugen eines neuen Objekts.

Kapitel 3 Aufbau eines SuM-Programms

In diesem Kapitel sollen Sie lernen:

- wie man ein SuM-Programm aus einer Vorlage erzeugt
- wie man in einem Programm Bibliotheksklassen zur Verfügung stellt
- wie man in einem Programm ein Objekt erzeugt
- wie man in einem Programm die Dienste eines Objekts benutzt
- wie man Objekte freigibt
- wie man ein Programm übersetzt und startet
- wie man ein Programm verändert
- wie man Fehlermeldungen beim Übersetzen auswertet

Nachdem Sie im letzten Kapitel Objekte "per Hand" erzeugt haben und ihre Dienste benutzt haben, sollen Sie in diesem Kapitel lernen, wie man diese Vorgänge automatisiert. Sie werden ein kleines Java-Programm schreiben, testen und verbessern. Dazu wird Ihnen eine Vorlage (Template) bereitgestellt, die die immer wieder benötigten Programmteile schon enthält. So müssen Sie nur die Vorlage an Ihre Wünsche anpassen.

3.1 Erzeugen einer Programm-Vorlage

Starten Sie BlueJ und erzeugen Sie ein neues Projekt mit dem Namen `Zwei`. Klicken Sie den Knopf `Neue Klasse`. Tragen Sie als Klassenname `MeinProgramm` ein und wählen Sie den Radioknopf `SuMKern` aus.

Abbildung 3.1: Erzeugung eines SuMKern-Programms

Bestätigen Sie mit Ok und im Projektfenster erscheint ein neues Symbol mit dem Namen `MeinProgramm`. Für Java ist auch ein Programm eine Klasse, deshalb beginnt `MeinProgramm` mit einem Großbuchstaben.

Abbildung 3.2:
Klassensymbol der
Klasse MeinProgramm

Das Symbol steht für die Klasse `MeinProgramm`. Die diagonale Schraffur bedeutet, dass diese Klasse noch übersetzt werden muss. Klicken Sie in den Knopf `Übersetzen` und die Schraffur verschwindet. Klicken Sie mit der rechten Maustaste (Mac ctrl-Klick) in das Symbol und ein Kontextmenü klappt auf. Wählen Sie `Bearbeiten`.

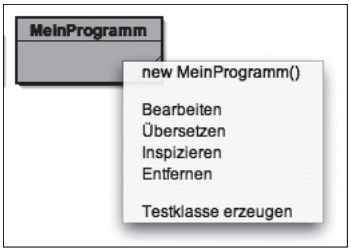


Abbildung 3.3:
Kontextmenü zur
Klasse MeinProgramm

Jetzt öffnet sich ein Fenster mit dem Programmtext.

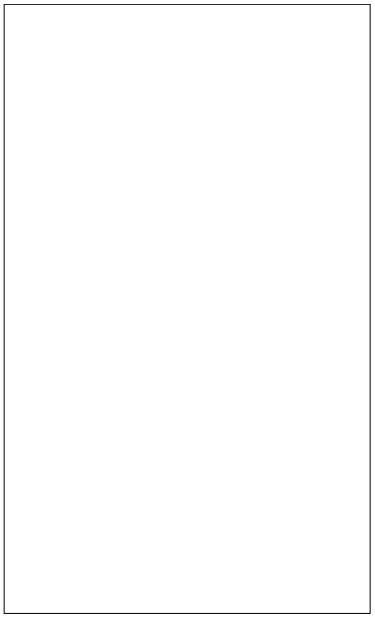


Abbildung 3.4:
Programmtext der
Klasse MeinProgramm

Dieses Fenster nennt man *Editor*. Der Editor dient dazu, Programmtext zu verändern. Im Prinzip ist der Editor eine einfache Textverarbeitung. Er kann aber auch mehr, z.B. werden bestimmte Worte wie `import` farbig hervorgehoben. In Kapitel 1 haben Sie in den Einstellungen `Zeilennummern anzeigen` ausgewählt, deshalb sehen Sie links die Zei-

lennummerierung.

Sie sehen, dass der Programmtext eine Struktur aufweist. Es gibt Einrückungen und Leerzeilen. Zu jedem Symbol geschweifte Klammer auf '{' gibt es das entsprechende Symbol '}'. Diese formale Struktur erhöht die Lesbarkeit eines Programms und dadurch wird es für eine andere Person einfacher, das Programm zu verstehen und eventuell zu verändern. Versuchen Sie von Beginn an, diese Struktur in Ihren Programmtexten zu erzeugen, die äußere Form ist ein nicht unwichtiges Bewertungskriterium für Programme.

3.2 Programmaufbau

Hinter das Wort `@author` sollten Sie Ihren Namen schreiben und hinter das Wort `@version` das aktuelle Datum. So kann man später gedruckte oder kopierte Texte einem Autoren zuordnen.

In Zeile 2 sehen Sie die sogenannte import-Anweisung. Sie sorgt dafür, dass im restlichen Programmtext die Klassen des Bibliothekspakets `sum.kern` zur Verfügung stehen. Das Sternsymbol * bedeutet dabei: alle Klassen des Pakets. Man hätte stattdessen auch zwei import-Anweisungen schreiben können:

```
import sum.kern.Bildschirm;
import sum.kern.Stift;
```

So werden nur bestimmte Klassen (Bildschirm und Stift) des Pakets importiert.

`public class MeinProgramm` bedeutet, dass `MeinProgramm` eine Klasse ist. Das Schlüsselwort `public` bedeutet, dass diese Klasse von anderen Stellen angesprochen werden kann, also öffentlich ist. Sie werden später den Dienst `fuehreAus` der Klasse `SuMProgramm` aufrufen. Die beiden geschweiften Klammern in Zeile 8 und 31 geben an, wo die Klassenbeschreibung beginnt und endet.

Jede Klasse besitzt einen *Konstruktor*. Das ist ein besonderer Dienst, der aufgerufen wird, wenn ein Objekt dieser Klasse erzeugt wird. Im Konstruktor der Klasse `MeinProgramm` werden die beiden Objekte `derBildschirm` und `meinStift` erzeugt. Man könnte den Konstruktor auch als *Initialisierungsteil* bezeichnen. Der Konstruktor ist öffentlich (`public`) und besitzt den gleichen Bezeichner wie die Klasse. Mehr über Konstruktoren werden Sie in Kapitel 6 erfahren.

Die Klasse `MeinProgramm` besitzt nur einen Dienst, den Auftrag `fuehreAus`. Auch dieser Dienst ist mit `public` als öffentlich markiert. Das Wort `void` bedeutet, dass es sich hier um einen Auftrag handelt. Die beiden geschweiften Klammern in Zeile 22 und 30 zeigen den Anfang und das Ende des Dienstes an.

Die beiden Schrägstriche in den Zeilen 9, 13, 20, 23 und 27 bedeuten, dass der Rest der Zeile ein Kommentar ist, der dem Leser das Verständnis des Programmtexts erleichtern soll. Allgemein gilt: **Sparen Sie nicht an Kommentaren!**

In Zeile 10 und 11 werden zwei Objekte deklariert. Dabei wird zuerst der Klassenname und danach der Objektname angegeben.

Im Konstruktor (Initialisierungsteil) werden die beiden Objekte erzeugt. Es wird jeweils

ein Konstruktor der entsprechenden Klasse mit dem Schlüsselwort `new` aufgerufen. Das Gleichheitszeichen "=" nennt man den *Zuweisungsoperator* und man liest ihn als *wird zu* (nicht gleich). Die mit `new` erzeugten Objekte werden also mit dem Zuweisungsoperator den entsprechenden Objektbezeichnern zugewiesen. Beachten Sie die Semikolons am Ende der Zeilen.

Im Aktionsteil werden dem Objekt `meinStift` zwei Aufträge geschickt (in Punktnotation). Beachten Sie die Anführungszeichen in Zeile 25. Einen solchen Parameter bezeichnet man als Zeichenkette bzw. `String`. Beachten Sie auch hier die Semikolons am Ende der Zeilen.

Guter Stil ist es, am Ende des Programms den benutzten Speicherbereich wieder freizugeben. Man schickt den Objekten die Anweisung `gibFrei()`.

Der Auftrag `derBildschirm.gibFrei()` sollte immer die **letzte** Anweisung in einem SuM-Kern-Programm sein. Sie sorgt dafür, dass das Bildschirmfenster stehen bleibt und als Titel *Das SuM-Programm ist beendet* erhält. Bei einem Klick auf das Fenster verschwindet es dann. Falls man diese Anweisung vergisst, verschwindet das Fenster sofort und Sie können sich die Zeichnung nicht mehr ansehen. Beachten Sie die Semikolons am Ende der Zeilen.

3.3 Programmausführung

Klicken Sie in den Knopf `Übersetzen` des Editors und das Programm wird erneut in den sogenannten Java-Bytecode übersetzt. Ein übersetztes Programm kann vom Rechner ausgeführt werden. Dazu schließen Sie den Editor und klicken mit der rechten Maustaste (Mac ctrl-Klick) auf das Klassensymbol `MeinProgramm`. Wählen Sie `new MeinProgramm()`. Es erscheint ein Dialog, um den Namen des neuen Objekts der Klasse `MeinProgramm` einzugeben. Benutzen Sie den vorgeschlagenen Namen `meinProg1`. Sie sehen unten in der Werkbank (engl. work bench), dass Sie ein neues Objekt erzeugt haben.

Abbildung 3.5:
Aufruf des Dienstes fuehreAus

Mit einem Rechtsklick auf dieses Objekt können Sie den Dienst `fuehreAus` aufrufen. Nach kurzer Zeit öffnet sich das Fenster `SuM-Programm`, das den Text "Hallo Welt" enthält. Wenn Sie in das Fenster klicken, verschwindet es wieder und auch das Objekt wird automatisch aus der Werkbank entfernt.

3.4 Fehlersuche

Ein großes Problem für Programmieranfänger in Java ist die Fehlersuche. Auch hierbei wird man von BlueJ sehr gut unterstützt. Dies sollen Sie an mehreren absichtlich erzeugten Fehlern kennenlernen.

Öffnen Sie dazu den Editor mit einem Doppelklick auf das Klassensymbol `MeinPro-`

gramm oder klicken Sie mit der rechten Maustaste (Mac ctrl-Klick) in das Klassensymbol und wählen Sie Bearbeiten. Ändern Sie in Zeile 2 das Wort import in Import. Java unterscheidet zwischen Groß- und Kleinschrift (im Gegensatz zu Pascal). Haben Sie bemerkt, dass die Farbe der Schrift sich geändert hat? Import ist kein Schlüsselwort für Java. Klicken Sie anschließend in den Knopf Übersetzen. Sie erhalten unten im Editor die Meldung:

Abbildung 3.6: Fehlermeldung beim Übersetzen

Diese Fehlermeldung bedeutet, das Schlüsselwort class oder interface wurde erwartet, aber Import wurde vorgefunden. Die Fehlermeldung ist knapp und nicht sehr aussagekräftig.

Klicken Sie jetzt in das Fragezeichen unten rechts im Editorfenster und Sie erhalten eine umfangreichere deutsche Fehlerbeschreibung:

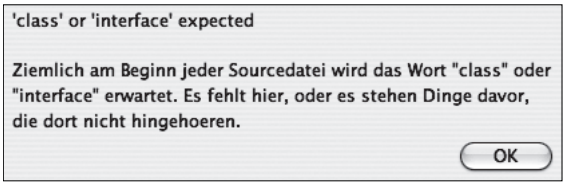


Abbildung 3.7: Ausführliche Fehlermeldung

Leider weist diese Meldung auch noch nicht auf den falschen Großbuchstaben hin, ist aber schon bedeutend aufschlussreicher. In den folgenden Übungen sollen Sie die Fehlermeldungen für anfängertypische Fehler untersuchen:

Übung 3.1 Ändern Sie sum.kern.* in sum.kerne.* und rufen Sie die ausführliche Fehlermeldung auf.

Übung 3.2 Entfernen Sie das Semikolon hinter sum.kern.* und rufen Sie die ausführliche Fehlermeldung auf.

Übung 3.3 Entfernen Sie eine geschweifte Klammer '{' und rufen Sie die ausführliche Fehlermeldung auf.

Übung 3.4 Entfernen Sie eine geschweifte Klammer '}' und rufen Sie die ausführliche Fehlermeldung auf.

Wenn Sie Dienste in ihrer Schreibweise verändern, z.B. gibFrei in gibtFrei, sehen Sie, dass es auch für bestimmte Fehler (noch) keine ausführlichen Fehlererläuterungen gibt.

Hinweis: Prüfen Sie bei Fehlermeldungen zuerst, ob Sie die Worte richtig geschrieben haben. Oft ist die Groß- Kleinschrift falsch, Worte werden getrennt statt zusammen geschrieben (gibFrei ist korrekt, gib Frei ist falsch). Oft wird ein Semikolon am Ende einer Zeile vergessen oder fälschlicherweise geschrieben.

Regel: Vor dem Zeichen '{' und nach dem Zeichen '}' darf **kein** Semikolon stehen.

3.5 Programmgesteuertes Zeichnen

Jetzt sollen Sie das vorgegebene Programm verändern und testen.

Übung 3.5 Ändern Sie den Programmtext so ab, dass die Zeichnung aus Abbildung 3.8 im Bildschirmfenster entsteht.

Hinweis: Versuchen Sie schrittweise vorzugehen und testen Sie zwischendurch die Ergebnisse. Vergessen Sie nicht den Stift hoch oder runter zu setzen. Sparen Sie nicht an Kommentaren!!!

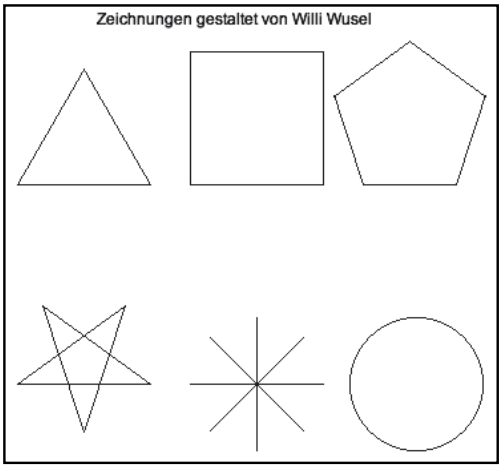


Abbildung 3.8: Programmgesteuertes Zeichnen

Hinweis: Manchmal passiert es, dass Sie das Programm erneut übersetzen wollen, während es noch im Hintergrund läuft. Sie können ein laufendes Programm immer beenden, indem Sie mit der rechten Maustaste (Mac ctrl-Klick) auf den Drehbalken (Barberpole) unten links im Projektfenster klicken.

Abbildung 3.9: Abbrechen eines laufenden Programms

3.6 Zusammenfassung

In diesem Kapitel haben Sie den Aufbau eines einfachen SuM-Kern-Programms kennen gelernt. Zu Beginn des Programm sollte der Autor / die Autoren und das Datum eingetragen werden. Die Bibliothek `sum.kern` muss importiert werden, * bedeutet dabei, dass alle Klassen des Pakets `sum.kern` importiert werden. Danach wird die Klasse `MeinProgramm` definiert. Die Klassendefinition endet mit der letzten Zeile 31, wie man an der geschweiften Klammer '}' erkennt. In dieser Klasse werden zuerst die sogenannten

Bezugsobjekte deklariert. Die Klasse enthält neben dem Konstruktor einen einzigen Dienst `fuehreAus`. Der Konstruktor enthält den Initialisierungsteil des Programms. Der Dienst `fuehreAus` enthält die beiden weiteren Teile eines jeden Programms: *Aktionsteil* und *Aufräumen*. Kommentare erkennen Sie an '//'. Bei den einzelnen Anweisungen wird die Punktnotation benutzt. Am Ende jeder Zeile folgt ein Semikolon ';'. Vor '{' und nach '}' darf kein Semikolon stehen. Die letzte Programmanweisung ist immer `derBildschirm.gib-Frei()`.

Sie haben auch gelernt, wie man ein Programm verändert, übersetzt, Fehler sucht, das Programm ausführt und zur Not abbricht. Vor der Programmausführung wird ein Programm übersetzt. Zuerst wird mit `new MeinProgramm()` ein Objekt der Klasse `MeinProgramm` erzeugt. Das Programm wird dann mit dem Aufruf des Dienstes `fuehreAus` des Objekts gestartet.

Programme kann man zur Not mit einem Rechtsklick auf den *Barberpole* abbrechen.

Programme kann man im Editorfenster verändern. Der Editor wird mit einem Rechtsklick oder einem Doppelklick auf das Klassensymbol im Projektfenster geöffnet.

Beim Übersetzen eines Programms werden eventuell Fehlermeldungen unten im Editor angezeigt. Durch Klick in das Fragezeichen erhält man eine ausführliche Fehlermeldung.

Neue Begriffe in diesem Kapitel

• **Programm-Vorlage (Template)** Viele Anweisungen kommen in jedem Programm vor. Deshalb bietet BlueJ die Möglichkeit diesen Text in eine Vorlage zu schreiben, die dann automatisch erzeugt wird. Dadurch vermeidet man Fehler und muss nicht soviel schreiben.

• **Übersetzen** Ein Javaprogramm kann erst ausgeführt werden, wenn es in den sogenannten Bytecode (für uns nicht lesbar) übersetzt wurde. Dieser Bytecode wird dann von einem Programm, der sogenannten Virtuellen Maschine (VM) ausgeführt.

• **Editor** Spezielles Textverarbeitungsprogramm mit Unterstützung der Javasyntax durch Färbung bestimmter Worte. Im Editor schreibt und ändert man Programme.

• **Einrücken** Allgemein übliche Methode zur optischen Verdeutlichung der Programmstruktur. Sollte unbedingt eingehalten werden.
• **Kommentar** Eine Kommentarzeile beginnt mit einem Doppelslash '//'. Kommentare werden bei der Programmübersetzung ignoriert und dienen nur der besseren Lesbarkeit. Sparen Sie nicht an Kommentaren!

• **Fehlermeldung** Beim Übersetzen werden oft Schreib- und Syntaxfehler erkannt. BlueJ versucht die passenden Fehlermeldungen auszugeben. Es gibt die Möglichkeit, Fehlermeldungen ausführlich anzuzeigen.

• **Barberpole** Balken unten links im Projektfenster. Wenn er rot schraffiert ist, zeigt er an, dass gerade ein Programm ausgeführt oder übersetzt wird. In Amerika war ein senkrechter Barberpole auf der Straße das Symbol für einen Friseurladen (barber shop).

Java-Bezeichner

• **import** Anweisung, um Klassen zur Verfügung zu stellen, die in einer Bibliothek im Bytecode zur Verfügung stehen. Der Quelltext ist dazu nicht notwendig.

• **public** öffentlich, Markierung für Klassen oder Dienste, die von "außen" benutzt werden sollen.

• **class** Schlüsselwort für die Deklaration einer Klasse.

• **new** Schlüsselwort zur Erzeugung eines neuen Objekts. Dahinter muss der Konstruktor einer Klasse aufgerufen werden.

• **'='** Zuweisung, wird als *wird zu* ausgesprochen. Der Bezeichner links erhält den Wert des Ausdrucks rechts.

Kapitel 4 Kontrollstrukturen I

In diesem Kapitel sollen Sie lernen:

- wie man den Programmfluss mit Kontrollstrukturen steuert
- wie man Kontrollstrukturen in Struktogrammen darstellt
- wie man Struktogramme verschachtelt
- was man unter einer ist-Beziehung versteht
- wie man in Java Schleifen programmiert
- wie man in Java Verzweigungen programmiert

Ein Programm besteht zum großen Teil aus Anweisungen, d.h. Aufrufen von Diensten, die Klassen zur Verfügung stellen. Diese Anweisungen werden der Reihe nach ausgeführt. Oft ist es aber so, dass bestimmte Anweisungen mehrmals ausgeführt werden müssen. Dazu bieten alle Programmiersprachen ein oder mehrere Konstrukte, die *Schleifen* (engl. Loop), an. Manchmal ist es auch notwendig, abhängig von bestimmten Bedingungen, unterschiedliche Anweisungen auszuführen. Ein solches Konstrukt nennt man eine *Verzweigung* oder *Auswahl*.

In diesem Kapitel sollen Sie an einem Beispielprojekt diese Java-Konstrukte, man nennt sie Kontrollstrukturen, kennen lernen. Sie sollen schrittweise ein Malprogramm entwickeln, so dass Sie mit der Maus auf dem Bildschirm zeichnen können. Dabei wird die Aufgabenstellung mehrfach abgewandelt, um die verschiedenen Kontrollstrukturen einzuführen. Das Projekt heisst *Freihandzeichnen*.

Zu den verschiedenen Kontrollstrukturen sollen Sie grafische Darstellungen kennen lernen, die sogenannten *Struktogramme*.

4.1 Schleife mit Ausgangsbedingung

Problemstellung 1: Malen durch Bewegung der Maus
Wenn die Maus bewegt wird, so soll eine Linie bzw. Kurve dem Mauszeiger (engl. Cursor) folgen. Das Programm soll beendet werden, wenn die Maus gedrückt wird.

Sie benötigen also den Bildschirm, einen Stift und ein Objekt, das die Maus darstellt, ein Objekt der Klasse `Maus`.

Übung 4.1 Öffnen Sie im Hilfemenü die Dokumentation `sum.kern` und wechseln Sie zur Klasse `Maus`. Wie sieht der Konstruktor aus? Welche Dienste bietet die Klasse an? Welche Dienste sind Anfragen, welche Dienste sind Aufträge? Notieren Sie sich die Dienste, Sie werden sie später benötigen.

Die benötigten Objekte sollen `derBildschirm`, `dieMaus` und `meinStift` heißen. Sie

müssen in der Klasse `MeinProgramm` deklariert werden, im Konstruktor erzeugt werden und beim Aufräumen freigegeben werden. Übrig bleibt der Aktionsteil im Dienst `fuehreAus`.

Der Artikel *der* bzw. *die* bei den beiden Objekten `derStift` bzw. `dieMaus` soll andeuten, dass es von diesen Objekten nur jeweils ein Exemplar geben kann. Der Artikel *mein* beim Objekt `meinStift` zeigt an, dass es in einer Klasse mehrere Stifte geben kann.

Die Maus soll dem Cursor folgen, solange die Maus nicht gedrückt ist.
Strukturiert geschrieben sieht das so aus:

```
wiederhole
    bewege den Stift zur Mausposition
solange die Maus nicht gedrückt ist
```

Diese Schreibweise nennt man *Pseudocode*. In Java sieht das dann so aus:

```
do
{
    meinStift.bewegeBis(dieMaus.hPosition(), dieMaus.vPosition());
} while (!dieMaus.istGedrueckt());
```

Das Ausrufezeichen '!' zu Beginn der Klammer hinter `while` bedeutet *nicht*. Beachten Sie die Einrückung. So wird das Innere der Schleife gekennzeichnet. Da die Durchlaufbedingung am Ende der Schleife steht, nennt man diese Kontrollstruktur *Schleife mit Ausgangsbedingung*. Die grafische Darstellung für diese Kontrollstruktur, das *Struktogramm*, sieht so aus:

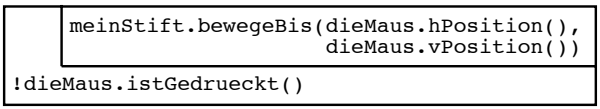


Abbildung 4.1: Struktogramm zur Schleife mit Ausgangsbedingung

Dabei kann der Text im Struktogramm auch im Pseudocode formuliert sein.

Beachten Sie: Die Bedingung ist eine *Durchlaufbedingung*. Solange die Bedingung erfüllt ist, wird die Schleife erneut durchlaufen.

Bevor jedoch diese Schleife durchlaufen wird, bewegen Sie den Stift zur aktuellen Mausposition und senken Sie die Maus ab. Jetzt kann die Schleife durchlaufen, also mit der Maus gezeichnet werden. Das Struktogramm sieht dann so aus:

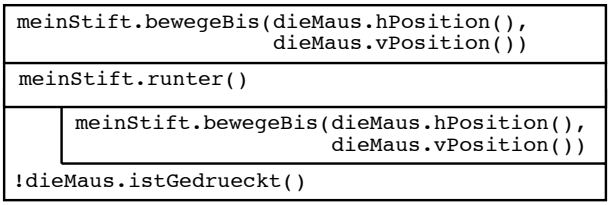


Abbildung 4.2: Struktogramm zur Sequenz und Schleife mit Ausgangsbedingung

Wenn mehrere Anweisungen hintereinander folgen, wie die beiden ersten Aufträge an

den Stift, so nennt man diese Struktur eine *Sequenz*. Oft werden die Anweisungen einer Sequenz mit geschweiften Klammern "{}" zusammengefasst.

Man kann die Schleife wie eine einzige Anweisung auffassen und hat somit eine Sequenz aus drei Anweisungen, deren dritte eine Schleife mit Ausgangsbedingung ist, deren Inneres aus einer Anweisung besteht.

Übung 4.2 Erzeugen Sie ein neues Projekt mit dem Titel `Freihand1`. Erzeugen Sie ein neues SuM-Kern-Programm und ändern Sie es so ab, dass die Problemstellung 1 gelöst wird.

4.2 Einseitige Verzweigung

Speichern Sie Ihr Projekt jetzt als `Freihand2`. So bleibt Ihre alte Lösung erhalten und Sie können das vorhandene Programm trotzdem verändern. Die Problemstellung soll jetzt geändert werden.

Problemstellung 2: Punkte zeichnen
Wenn die Maus gedrückt wird, so soll an der Mausposition ein Punkt (kleiner Kreis) gezeichnet werden. Wenn die Maus in gedrücktem Zustand bewegt wird, soll also eine Kurve von Punkten entstehen. Das Programm soll mit einem Doppelklick beendet werden, da der Mausdruck schon anders verwendet wird.

Man erkennt sofort, dass auch hier eine Schleife mit Ausgangsbedingung verwendet werden kann. Allerdings werden im Schleifeninneren kleine Kreise gezeichnet. Diese Kreise sollen aber nur gezeichnet werden, wenn die Maustaste gedrückt ist. Sie benötigen hier eine *bedingte Anweisung* oder auch Selektion genannt.

In Pseudocode:

```
wiederhole
    wenn die Maus gedrückt ist
        bewege den Stift zur Mausposition
        zeichne einen kleinen Kreis
solange die Maus nicht doppelt geklickt worden ist
```

in Java:

```
do
{
  if (dieMaus.istGedrueckt())
  {
     meinStift.bewegeBis(dieMaus.hPosition(),dieMaus.vPosition());
     meinStift.zeichneKreis(3);
  }
} while (!dieMaus.doppelKlick());
```

Falls die Verzweigung nur eine einzige Anweisung enthält, können die geschweiften Klammern entfallen.

Das Struktogramm sieht dann so aus:

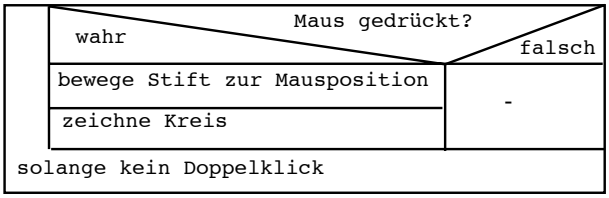


Abbildung 4.3: Struktogramm zur einseitigen Verzweigung

Übung 4.3 Ändern Sie das SuM-Kern-Programm so ab, dass die Problemstellung 2 gelöst wird.

4.3 Schleife mit Eingangsbedingung

Speichern Sie Ihr Projekt jetzt als `Freihand3`. So bleibt Ihre alte Lösung erhalten und Sie können das vorhandene Programm trotzdem verändern. Die Problemstellung soll jetzt geändert werden.

Problemstellung 3: Einzelne Punkte zeichnen
Wenn die Maus gedrückt wird, so soll an der Mausposition ein Punkt (kleiner Kreis) gezeichnet werden. Ein neuer Punkt soll allerdings erst dann gezeichnet werden, wenn die Maus erneut gedrückt wird. Das Programm soll mit einem Doppelklick beendet werden.

Wenn ein Punkt gezeichnet wurde, muss das Programm also warten, bis die Maus losgelassen wurde. Dies erreicht man durch eine so genannte Warteschleife.

In Pseudocode:

```
solange die Maus gedrückt ist
    tue nichts
```

in Java:

```
while (dieMaus.istGedrueckt())
    ; // tue nichts  <= Kommentar
```

Hier steht die Durchlaufbedingung am Anfang. Das Schleifeninnere ist leer. Zur Verdeutlichung ist aber ein Kommentar ergänzt. Dies nennt man eine *Schleife mit Eingangsbedingung*:

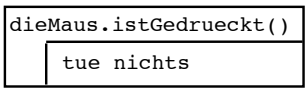


Abbildung 4.4: Struktogramm zur Schleife mit Eingangsbedingung

Übung 4.4 Ändern Sie das SuM-Kern-Programm so ab, dass die Problemstellung 3 gelöst wird. Zeichnen Sie das vollständige Struktogramm dazu.

4.4 Zweiseitige Verzweigung

Die zweiseitige Verzweigung wird auch zweiseitige Auswahl genannt. Sie ist eine Erweiterung der einseitigen Verzweigung. Die Problemstellung wird jetzt so geändert, dass das zu Beginn dieses Kapitels geplante Freihandzeichnen endlich so funktioniert, wie Sie es aus einfachen Malprogrammen kennen.

Problemstellung 4: Freihandzeichnen
Wenn die Maus gedrückt ist soll mit der Maus eine Kurve gezeichnet werden, wenn die Maustaste losgelassen ist, soll nicht gezeichnet werden. Das Programm soll mit einem Doppelklick beendet werden.

Hier soll nur die zweiseitige Verzweigung dargestellt werden. Sie sollen sie dann in das Programm einbauen. Vergessen Sie nicht, vorher das Projekt in `Freihand4` umzubenennen.

Im Pseudocode:

```
wenn Bedingung erfüllt
    Anweisung 1
sonst
    Anweisung 2
```

In Java:

```
if (Bedingung)
    Anweisung1;
else
    Anweisung2;
```

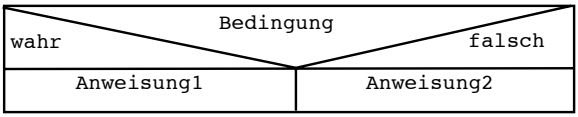


Abbildung 4.5: Struktogramm zur zweiseitigen Verzweigung

Übung 4.5 Ändern Sie das SuM-Kern-Programm so ab, dass die Problemstellung 4 gelöst wird. Zeichnen Sie das vollständige Struktogramm dazu. Versuchen Sie, die zweiseitige Verzweigung einzubauen. Es gibt auch Lösungsmöglichkeiten mit einseitiger Verzweigung.

4.5 Klasse Tastatur

Für die nächste Problemstellung benötigen Sie die Klasse `Tastatur`. Es wird keine weitere Kontrollstruktur eingeführt.

Übung 4.6 Öffnen Sie im Hilfemenü die Dokumentation `sum.kern` und wechseln Sie zur Klasse `Tastatur`. Wie sieht der Konstruktor aus? Welche Dienste bietet die Klasse an? Welche Dienste sind Anfragen, welche Dienste sind Aufträge? Notieren Sie sich die Dienste, Sie werden sie später benötigen.

Problemstellung 5: Auf Tastendruck vom Zeichnen zum Radieren wechseln
Das Programm soll so erweitert werden, dass bei einem Tastendruck der Stift vom Zeichenmodus (Normalmodus) in den Radiermodus aber nicht wieder zurück wechselt.

Ein Stift kann mit dem Auftrag `radiere()` in den Radiermodus umgeschaltet werden. Mit dem Auftrag `normal()` wird dann wieder in den Zeichenmodus umgeschaltet. Beachten Sie, dass der Stift nur radiert, wenn er vorher gesenkt wurde.

Hier sollen Sie üben, zu einem vorgegebenen Struktogramm das Programm zu schreiben. Wechseln Sie zu `Freihand5`!

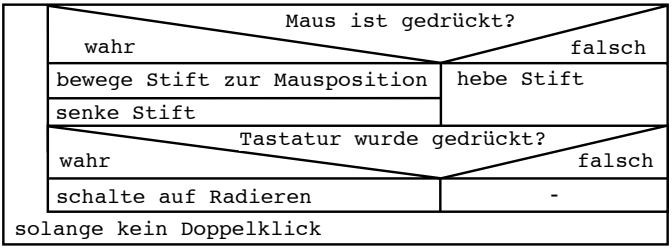


Abbildung 4.6: Struktogramm

Übung 4.7 Ändern Sie das SuM-Kern-Programm ab, indem Sie das Struktogramm aus Abbildung 4.6 in Java-Anweisungen umsetzen.

4.6 Linien zeichnen

Auch in diesem Abschnitt soll keine neue Kontrollstruktur eingeführt werden, die bereits bekannten Kontrollstrukturen sollen geübt werden. Erzeugen Sie das Projekt `Freihand6`. Auch hier soll das zuletzt erstellte Programm abgeändert werden.

Problemstellung 6: Linien
Bei einem Druck des Mausknopfs wird an der Mausposition ein Punkt gezeichnet. Dieser wird mit einer geraden Linie mit der Mausposition im Moment des Loslassens des Mausknopfs verbunden. Das Programm soll ansonsten wieder mit einem Doppelklick beendet werden.

Übung 4.8 Ändern Sie das SuM-Kern-Programm und lösen Sie Problemstellung 6.

4.7 ist-Beziehung

In diesem Abschnitt sollen Sie die Klasse `Buntstift` benutzen. Ein Buntstift kennt alle Dienste der Klasse `Stift` und dazu eine Reihe zusätzlicher Dienste. Man kann also sagen, der Buntstift ist ein Stift, der zusätzliche Dienste besitzt. Man nennt eine solche Beziehung **ist-Beziehung**. Ein Buntstift **ist** ein Stift. Solche Beziehungen gibt es im täglichen Leben häufig: Ein Auto ist ein Fahrzeug, ein PKW ist ein Auto, ein VW-Golf ist ein PKW, ein Golf IV ist ein Golf. Auch hier findet eine fortlaufende Spezialisierung statt.

In der objektorientierten Programmierung benutzt man dafür die Begriffe **Oberklasse** und **Unterklasse**. Stift ist die Oberklasse, Buntstift die Unterklasse. Oberklassen sind allgemeiner, Unterklassen sind spezieller. Oberklassen können weniger, Unterklassen können mehr. Unterklassen können alles, was auch die Oberklasse kann. In der Unterklasse kommen zusätzliche Dienste hinzu. Man sagt, die Unterklasse **erbt** alle Dienste der

Oberklasse. Die Dienste einer Unterklasse lassen sich in drei Kategorien einteilen:

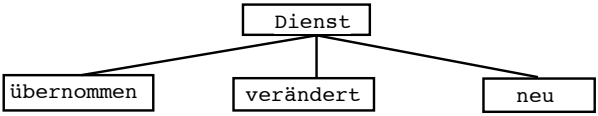


Abbildung 4.7: Dienste der Unterklassen

Übung 4.9 Öffnen Sie im Hilfemenü die Dokumentation `sum.kern` und wechseln Sie zur Klasse `Buntstift`. Wie sieht der Konstruktor aus? Welche Dienste bietet die Klasse an? Welche Dienste sind Anfragen, welche Dienste sind Aufträge? Ordnen Sie die Dienste nach den Kategorien übernommen, verändert, neu.

Die Gliederung nach Ober- und Unterklassen ist in Java ein durchgängiges Konzept. Alle Javaklassen stammen entweder direkt oder mit zwischengeschalteten Klassen von der Klasse `Object` ab. In der Dokumentation können Sie auf die Darstellung `Tree` wechseln und sich den Klassenbaum grafisch darstellen lassen.

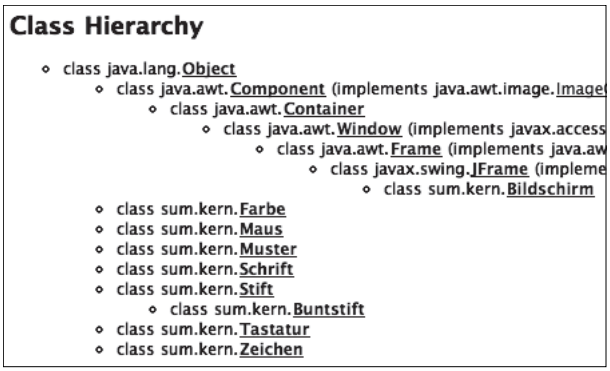


Abbildung 4.8: Hierarchie der SuM-Kern-Bibliothek

Mit etwas Fantasie kann man die Darstellung in Abbildung 4.8 als Baum auffassen. Die Wurzel ist oben links die Klasse `Object`. Die Klasse `Stift` ist direkt von der Klasse `Object` abgeleitet und die Klasse `Buntstift` ist wiederum von der Klasse `Stift` abgeleitet. Die Klasse `Bildschirm` hat einen besonders langen Weg bis zur Klasse `Object`. Im Kapitel 6 werden Sie sich intensiv mit Ober- und Unterklassen beschäftigen.

Problemstellung 7: Farbwechsel
Beim Freihandzeichnen aus Abschnitt 4.5 soll bei einem Tastendruck statt auf Radieren auf die Farbe Rot umgeschaltet werden.

Öffnen Sie das Projekt `Freihand5` und speichern Sie es als `Freihand7`. Um farbig zeichnen zu können muss der Stift durch einen Buntstift ersetzt werden. Der Objektbezeichner `meinStift` kann erhalten bleiben. Die Änderung betrifft also nur die Deklaration und Erzeugung des Objekts `meinStift`.

Um die Farbe des Stifts auf Rot zu ändern benutzen Sie den Auftrag

```
meinStift.setzeFarbe(Farbe.ROT);
```

Das Wort ROT ist in Großbuchstaben geschrieben, da es sich hier um eine Konstante handelt. Konstanten werden in Java üblicherweise in Großbuchstaben geschrieben. Die anderen Farbkonstanten können Sie sich in der Dokumentation zur Klasse `Farbe` ansehen. Auch die Klasse `Muster` enthält Konstanten für die Füllmuster von Rechtecken und Kreisen.

Übung 4.10 Ändern Sie das Programm so ab, dass bei Tastendruck auf die Farbe Rot umgeschaltet wird.

4.8 Tastaturpuffer

In diesem Abschnitt sollen Sie lernen, wie man mit der Tastatur wieder auf Schwarz zurückschalten kann. Dazu ist es notwendig, dass Sie sich die Klasse `Tastatur` noch einmal etwas genauer ansehen.

Problemstellung 8: Doppelter Farbwechsel
Beim Freihandzeichnen aus Abschnitt 4.7 soll bei einem Tastendruck die Farbe geändert werden und zwar bei der Taste 'r' auf Rot sonst auf Schwarz.

Bis jetzt haben Sie nur die Anfrage `dieTastatur.wurdeGedrueckt()` benutzt. Es gibt aber auch die Anfrage `dieTastatur.zeichen()`, die das gedrückte Zeichen liefert. Der Datentyp dazu heißt in Java `char` (von engl. character).

Eine Tastatur ist in Wirklichkeit etwas komplexer, als man zuerst denkt. Die Tastatur benötigt nämlich ein Gedächtnis, den sogenannten *Tastaturpuffer*, in dem sie sich die gedrückten Tasten merkt. Sobald das Betriebssystem ein Zeichen anfordert, wird das zuerst eingegebene Zeichen übergeben. Anschließend wird das Zeichen aus dem Puffer gelöscht. Den Tastaturpuffer kann man sich wie eine Warteschlange an einem Postschalter vorstellen. Nur warten nicht Personen sondern Zeichen.

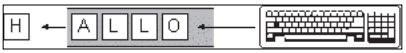


Abbildung 4.9:
Tastaturpuffer

Die Anfrage `dieTastatur.wurdeGedrueckt()` liefert den Wert wahr, wenn ein Zeichen im Tastaturpuffer steht, d.h. die Warteschlange mindestens ein Element enthält.

Die Anfrage `dieTastatur.zeichen()` liefert das erste Zeichen des Puffers. Dabei wird das Zeichen **nicht** aus dem Puffer entfernt. So ist es möglich, das Zeichen mehrfach auszulesen, ohne dass es verloren geht.

Der Auftrag `dieTastatur.weiter()` entfernt das vorderste Zeichen aus dem Puffer. Der Puffer ist anschließend entweder leer und die Anfrage `dieTastatur.wurdeGedrueckt()` liefert den Wert `false`, oder das nächste Zeichen steht vorne in der Warteschlange und kann mit `dieTastatur.zeichen()` ausgelesen werden.

Wichtig: Die Anfrage `zeichen()` und der Auftrag `weiter()` machen nur Sinn, wenn mindestens ein Zeichen im Tastaturpuffer ist. Ansonsten erscheint eine Fehlermeldung und das Programm wird beendet. Vor der Benutzung dieser beiden Dienste muss also dieTastatur mit der Anfrage `dieTastatur.wurdeGedrueckt()` getestet werden.

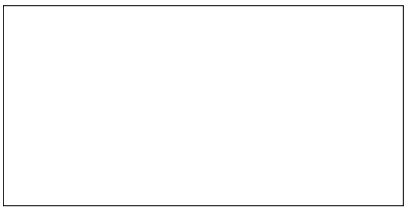


Abbildung 4.10:
Tastaturfehler

Eine Tastaturabfrage sieht im Pseudocode also typischerweise so aus:

```
wenn die Tastatur gedrückt wurde
    verarbeite das Zeichen
    entferne das Zeichen aus dem Puffer
```

Wie soll das Zeichen verarbeitet werden? Wenn ein 'r' eingegeben wurde, soll auf Rot umgeschaltet werden, ansonsten soll auf Schwarz geschaltet werden. Ein typischer Fall für eine zweiseitige Verzweigung:

```
if (dieTastatur.wurdeGedrueckt)
{
    if (dieTastatur.zeichen() == 'r')
            meinStift.setzeFarbe(Farbe.ROT);
    else
            meinStift.setzeFarbe(Farbe.SCHWARZ);
    dieTastatur.weiter();
}
```

Ein Vergleich wird in Java mit zwei Gleichheitszeichen durchgeführt '==').
Konkrete Zeichen müssen in Java mit einem Hochkomma umrahmt sein, Zeichenketten (Strings) dagegen mit Anführungszeichen.

Übung 4.11 Zeichnen Sie das Struktogramm zum Javatext oben.

Übung 4.12 Schreiben Sie das Java-Programm zur Problemstellung 8.

4.9 Mehrseitige Verzweigung

Zum Schluss dieses Kapitels soll die mehrseitige Verzweigung, auch Mehrfachauswahl genannt, behandelt werden. Ein typischer Fall ist die Auswertung der Tastatur. Je nach Tastendruck soll eine andere Farbe eingestellt werden.

Problemstellung 9: Mehrfacher Farbwechsel
Beim Freihandzeichnen aus Abschnitt 4.8 soll bei einem Tastendruck die Farbe geändert werden und zwar bei der Taste 'r' auf Rot, 'g' auf Grün, 'b' auf Blau usw. Bei allen anderen Tasten soll die Farbe auf Schwarz gesetzt werden.

Der wesentliche Teil des Programms sieht in Pseudocode folgendermaßen aus:

```
wenn dieTastatur gedrückt wurde
    falls das Zeichen ist
         'r' : setze Stiftfarbe auf rot
         'b' : setze Stiftfarbe auf blau
```

```
        'g' : setze Stiftfarbe auf grün
       andernfalls : setze Stiftfarbe auf schwarz
  entferne erstes Zeichen aus dem Puffer
```

In Java ist die Syntax etwas ungewöhnlich. Das liegt daran, dass diese Syntax von der Programmiersprache C in Java übernommen worden ist.

```
if (dieTastatur.wurdeGedrueckt())
{
  switch (dieTastatur.zeichen())
  {
    case 'r': case 'R': meinStift.setzeFarbe(Farbe.ROT); break;
    case 'b': case 'B': meinStift.setzeFarbe(Farbe.BLAU); break;
    case 'g': case 'G': meinStift.setzeFarbe(Farbe.GRUEN); break;
    default: meinStift.setzeFarbe(Farbe.SCHWARZ); break;
  }
  dieTastatur.weiter();
}
```

Die Klammer hinter dem Wort `switch` (deutsch Schalter) enthält den Wert, der getestet wird. Hinter `case` (deutsch Fall) steht eine oder mehrere Auswahlen. Nach dem Doppelpunkt folgen die Anweisungen, die im entsprechenden Fall ausgeführt werden sollen. 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`-Anweisung zu Ende ist. An dieser Stelle ist Java sehr fehleranfällig. 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.

Das zugehörige Struktogramm sieht so aus:

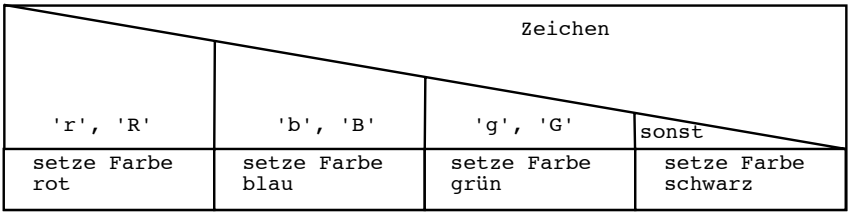


Abbildung 4.11:
Struktogramm zur mehrseitigen Verzweigung

Übung 4.13 Schreiben Sie das Programm zur Problemstellung 9. Ergänzen Sie weitere Farben. Sie finden die Farben in der Klassendokumentation zur Klasse `Farbe`.

Übung 4.14 Erweitern Sie das Programm so, dass mit den Tasten '1' bis '9' die Linienbreite verändert wird. Den zugehörigen Auftrag finden Sie in der Klassendokumentation zur Klasse `Buntstift`.

4.10 Zusammenfassung

In diesem Kapitel haben Sie folgende Kontrollstrukturen (in der Reihenfolge ihrer Einführung) kennen gelernt:

- die Schleife mit Ausgangsbedingung (do-Schleife)
- die einseitige Verzweigung (if-Anweisung)
- die Schleife mit Eingangsbedingung (while-Schleife)
- die zweiseitige Verzweigung (if-else-Anweisung)
- die mehrseitige Verzweigung (switch-Anweisung)

Der Form halber nimmt man die Sequenz, das Hintereinanderausführen von Anweisungen hinzu.

Sie haben gesehen, dass man diese Kontrollstrukturen auf drei Arten schreiben kann:

- als Pseudocode,
- als Java-Programmausschnitt,
- als Struktogramm.

Sie sollten jetzt in der Lage sein, die drei Darstellungsarten ineinander zu überführen.

Sie haben mit der Klasse `Buntstift` die ist-Beziehung zwischen Klassen kennen gelernt.

- Sie können die Begriffe Oberklasse- und Unterklasse richtig benutzen.
- Sie wissen, dass eine Unterklasse die Dienste der Oberklasse erbt (übernimmt), die Unterklasse kann diese Dienste aber auch abändern. In der Unterklasse können zusätzliche Dienste vorhanden sein.
- Die Oberklasse ist allgemeiner, die Unterklasse ist spezieller.

Sie haben die Klasse Tastatur kennen gelernt.

- Sie wissen wie ein Tastaturpuffer bedient wird und mit welchen Aufträgen und Anfragen man darauf zugreifen kann.

Neue Begriffe in diesem Kapitel

- **Kontrollstruktur** Sprachkonstrukt um den Programmfluss (die Reihenfolge der Anweisungen) zu steuern. Es gibt Schleifen und Verzweigungen.

- **Schleife** Wenn eine Reihe von Anweisungen mehrmals hintereinander ausgeführt werden soll, benutzt man dazu eine Schleife. Die Durchlaufbedingung kontrolliert, ob die Schleife noch mal durchlaufen werden soll. Wenn Sie am Anfang steht, wird die Schleife eventuell gar nicht durchlaufen (abweisende Schleife). Wenn sie am Ende steht, wird die Schleife mindestens einmal durchlaufen.

- **Durchlaufbedingung** Sie dient zur Kontrolle, ob eine Schleife noch mal durchlaufen werden muss. Die Bedingung muss einen Wahrheitswert (wahr bzw. falsch) liefern.

• **Verzweigung** Abhängig von einer Bedingung wird entschieden, welche Anweisungen ausgeführt werden. Es gibt einseitige, zweiseitige und mehrseitige Verzweigungen.

• **Struktogramm** Es dient dazu, eine Kontrollanweisung grafisch darzustellen. Struktogramme können verschachtelt werden, da ihre Form immer ein Rechteck ist. Der Text im Struktogramm kann in Java oder Pseudocode geschrieben sein.

• **Pseudocode** Wenn ein Programm(ausschnitt) in Umgangssprache und nicht in Java geschrieben ist, nennt man dies Pseudocode.

• **ist-Beziehung** Zwischen Klassen kann eine ist-Beziehung bestehen. Die eine Klasse nennt man dann Oberklasse, die andere Unterklasse. Oberklasse sind allgemeiner, Unterklassen sind spezieller.

• **Vererbung** Unterklassen erben die Dienste ihrer Oberklassen. Diese Dienste können aber auch angepasst werden.

• **Tastaturpuffer** Die in die Tastatur getippten Zeichen kommen in eine Warteschlange, den Tastaturpuffer. Sie bleiben im Puffer bis sie daraus entfernt werden.

Java-Bezeichner

• **while** (deutsch solange) leitet eine Durchlaufbedingung ein. Die Bedingung muss in Klammern stehen.

• **if** (deutsch wenn) steht vor der Bedingung zur ein- oder zweiseitigen Verzweigung. Die Bedingung muss eingeklammert werden.

• **else** steht vor dem Alternativteil der zweiseitigen Verzweigung. Eine einseitige Verzweigung hat keinen else-Teil.

• **switch** leitet eine Mehrfachverzweigung ein.

• **case** leitet einen Fall der Mehrfachverzweigung ein.

• **break** beendet einen Teil der Mehrfachverzweigung.

• **default** behandelt alle nicht mit `case` behandelten Fälle einer Mehrfachverzweigung.

• **'=='** ist der Vergleichsoperator. Er besteht aus zwei Gleichheitszeichen und wird als *gleich* ausgesprochen.

• **char** steht für Zeichen (engl. character). Zeichen werden zwischen Hochkommata gesetzt.

Kapitel 5 Kontrollstrukturen II

In diesem Kapitel sollen Sie lernen:

- wie man Programme schrittweise erweitert
- wie man zusammengesetzte Bedingungen bildet
- wie man eine Animation erzeugt
- wie man Zufallszahlen erzeugt
- wie man den Stift an eine zufällige Position bewegt
- wie man eine Stoppuhr erzeugt und benutzt

In diesem Kapitel sollen Sie üben:

- wie man in Java Schleifen programmiert
- wie man in Java Verzweigungen programmiert
- wie man Kontrollstrukturen in Struktogrammen darstellt

In diesem Kapitel sollen Sie zwei Spiele entwickeln, das Dartspiel und die Schatzsuche. Außerdem sollen mehrere kleine Programme mit Zufallszahlen erstellt werden.

Beim Dartspiel soll ein Dartpfeil auf eine Scheibe gelenkt werden. Das Spiel soll schrittweise aufgebaut werden. In den einzelnen Abschnitten dieses Kapitels wird das Spiel jeweils etwas erweitert. Dabei sollen Sie die Anwendung der Kontrollstrukturen aus Kapitel 4 üben.

5.1 Pfeil und Dartscheibe

Übung 5.1 Erzeugen Sie ein neues Projekt mit dem Titel `Dart1` und fügen Sie ein SuM-Kern-Programm ein. Der Bildschirm soll 600 * 400 Pixel groß sein.

Übung 5.2 Verändern Sie das Programm so, dass auf dem Bildschirm die Zeichnung aus Abbildung 5.1 entsteht. Zeichnen Sie die Dartscheibe zuerst und anschließend den Pfeil. Der Pfeil hat an der Spitze einen kleinen Kreis.

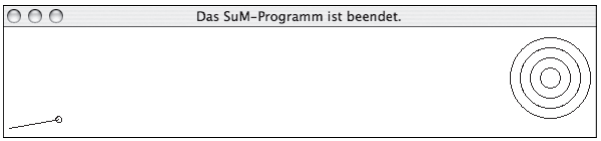


Abbildung 5.1: Bildschirm zum Dartspiel

5.2 Fliegender Pfeil

Für die nächste Aufgabe sollten Sie das Projekt in `Dart2` umbenennen.

Übung 5.3 Verändern Sie das Programm so, dass der Pfeil langsam in seine Richtung fliegt, bis seine Spitze eine gedachte vertikale Linie durch den Mittelpunkt der Scheibe erreicht.

Das Fliegen des Pfeils kann man durch wiederholtes Löschen und Neuzeichnen des Pfeils erreichen. Am einfachsten kann man den Pfeil löschen, indem man auf Radieren wechselt und die Zeichnung rückwärts ausführt. Der Auftrag `bewegeUm(double)` kann auch negative Zahlen als Parameter erhalten. Dann bewegt sich der Stift rückwärts.

Nach dem Löschen muss der Pfeil allerdings immer ein kleines Stück vorwärts bewegt werden. Die Länge dieses Stücks wirkt sich auf die Geschwindigkeit des Pfeils aus. Beachten Sie, dass man Dezimalzahlen z.B. 0.01 benutzen kann. Überlegen Sie sich, welche Kontrollstruktur sinnvoll ist.

Der Aktionsteil Ihres Programms besteht jetzt aus 3 Teilen:

```
// Dartscheibe zeichnen
…
// Pfeil zeichnen
…
// Pfeil fliegt
…
```

Der Pfeilflug muss die folgenden Pseudocode-Anweisungen solange ausführen, bis die gedachte vertikale Linie erreicht ist:

```
Lösche den Pfeil
Bewege den Stift um ein kleines Stück (in Pfeilrichtung)
Zeichne den Pfeil neu
```

5.3 Pfeil fällt

Für die nächste Aufgabe sollten Sie das Projekt in `Dart3` umbenennen. Jetzt erhält der Spieler die erste Eingriffsmöglichkeit in das Spiel.

Übung 5.4 Zu Beginn soll der Pfeil langsam nach unten fallen. Sobald die Maustaste gedrückt wird, soll der Pfeil losfliegen, bis er die gedachte Linie erreicht.

Die Klasse `Maus` wird benötigt. Die Struktur des Aktionsteils ändert sich:

```
// Dartscheibe zeichnen
…
// Pfeil zeichnen
…
// Pfeil fällt
…
// Pfeil fliegt
…
```

Welche Kontrollstrukturen benötigen Sie dazu?

5.4 Pfeil dreht

Vergessen Sie nicht, ihr Projekt umzubenennen. Der Spieler erhält jetzt die zusätzliche Möglichkeit, die Wurfrichtung zu verändern.

Übung 5.5 Zu Beginn soll der Pfeil nach unten fallen. Sobald die Maustaste gedrückt wird, soll der Pfeil sich langsam drehen. Wenn die Maustaste losgelassen wird, soll der Pfeil losfliegen, bis er die gedachte Linie erreicht.

Wie ändert sich die Struktur des Aktionsteils? Wie kann man prüfen, ob die Maus losgelassen wurde?

5.5 Treffer!

Vergessen Sie nicht, ihr Projekt umzubenennen. Jetzt soll überprüft werden, ob der Pfeil die Scheibe getroffen hat.

Übung 5.6 Zu Beginn soll der Pfeil nach unten fallen. Sobald die Maustaste gedrückt wird, soll der Pfeil sich langsam drehen. Wenn die Maustaste losgelassen wird, soll der Pfeil losfliegen, bis er die gedachte Linie erreicht. Falls die Scheibe getroffen wird, soll in der Bildschirmmitte das Wort *getroffen*, ansonsten das Wort *daneben* geschrieben werden.

Da der Pfeil an der gedachten Linie anhält, muss nur noch seine vertikale Position überprüft werden.

5.6 Wieder hoch!

Das Spiel hat eine ärgerliche Eigenschaft: Wenn der Pfeil zu weit fällt, ist er auf dem Bildschirm nicht mehr sichtbar. Das soll jetzt behoben werden.

Übung 5.7 Zu Beginn soll der Pfeil nach unten fallen. Falls er tiefer als der untere Bildschirmrand fällt, soll er wieder an den oberen Bildschirmrand versetzt werden. Sobald die Maustaste gedrückt wird, soll der Pfeil sich langsam drehen. Wenn die Maustaste losgelassen wird, soll der Pfeil losfliegen, bis er die gedachte Linie erreicht. Falls die Scheibe getroffen wird, soll in der Bildschirmmitte das Wort *getroffen*, ansonsten das Wort *daneben* geschrieben werden.

Versuchen Sie diese Aufgabe so zu lösen, dass Sie den Bildschirm nach seiner Höhe fragen. Testen Sie das Ergebnis, indem Sie die Bildschirmdimensionen (beim Erzeugen des Bildschirms) verändern. Machen Sie auch die Position der Dartscheibe und damit auch die gedachte Linie vom Bildschirm abhängig, d.h. ersetzen Sie die horizontale Position durch einen Ausdruck, in dem die Breite des Bildschirms benutzt wird.

5.7 Zwei Spieler

Jetzt soll ein zweiter Spieler beteiligt werden. Während der erste Spieler die Maus benutzt, soll der zweite Spieler die Tastatur benutzen.

Übung 5.8 Zu Beginn soll der Pfeil nach unten fallen. Falls er tiefer als der untere Bildschirmrand fällt, soll er wieder an den oberen Bildschirmrand versetzt werden. Sobald die Maustaste gedrückt wird, soll der Pfeil sich langsam drehen. Wenn die Maustaste losgelassen wird, soll der Pfeil losfliegen, bis er die gedachte Linie erreicht. Während dieses Fluges kann der zweite Spieler mit den Tasten 'l' und 'r' die Richtung des Pfeils um jeweils 5° verändern. Falls die Scheibe getroffen wird, soll in der Bildschirmmitte das Wort *getroffen*, ansonsten das Wort *daneben* geschrieben werden.

Dieses Spiel kann von den zwei Spielern mit- und gegeneinander gespielt werden. Auch wenn der Pfeil nach links abgeschossen wurde, kann der zweite Spieler noch das Ziel erreichen. Eventuell ist es sinnvoll, die Anfangsposition des Pfeils etwas nach rechts zu verlegen.

5.8 Jetzt wird's bunt

Durch den Einsatz von Farbe soll das Spiel etwas aufgepeppt werden.

Übung 5.9 Die Dartscheibe soll aus verschiedenfarbigen Ringen bestehen. Die Pfeilspitze soll ein roter Kreis sein.

Hier muss natürlich ein Buntstift eingesetzt werden. Einen ausgefüllten Kreis erhalten Sie, indem Sie vorher das Füllmuster auf `GEFUELLT` umstellen.

```
meinStift.setzeFuellmuster(Muster.GEFUELLT);
```

Die Klasse Muster besteht aus zwei Konstanten: `GEFUELLT` und `DURCHSICHTIG`.

Eine weitere Erweiterung wäre die Trefferunterscheidung *Mitte getroffen*, *erster Ring getroffen* usw.

Eine etwas anspruchsvollere Erweiterung bestünde aus der Möglichkeit, am Schluss zu entscheiden, ob noch mal gespielt werden soll.

Schauen Sie sich jetzt Ihr Programm noch mal kritisch an: Ihr Programmtext sollte durch Leerzeilen und Kommentare zwischen den einzelnen Teilaktionen strukturiert sein. Überprüfen Sie noch mal, ob eventuell andere Schleifen oder Verzweigungen die Verständlichkeit erhöhen könnten.

Übung 5.10 Zeichnen Sie das Struktogramm zum Dartspiel.

5.9 Zufallszahlen

In vielen Programmen werden Zufallszahlen benötigt. Ein Computer kann keine echten zufälligen Zahlen erzeugen. Ausgehend von einer Startzahl werden mit Hilfe einer Formel Zufallszahlen berechnet. Die jeweils nächste Zufallszahl wird also durch die Anwendung einer Formel auf die vorherige Zufallszahl berechnet. Als Startzahl wird meistens die Uhrzeit benutzt, so dass man nicht immer dieselbe Startzahl (und damit dieselbe Folge von Zufallszahlen) hat.

Hier bietet sich ein **mögliches Referatsthema** an: Verfahren zur Erzeugung von Zufallszahlen. Die notwendigen Informationen finden Sie im Internet.

Die SuM-Bibliothek stellt für Zufallszahlen das Paket `sum.werkzeuge` zur Verfügung, das eine Klasse `Rechner` enthält.

Übung 5.11 Rufen Sie im Hilfemenü von BlueJ `Dokumentation sum.werkzeuge` auf und informieren Sie sich über die Klasse `Rechner`. Welche Dienste zur Erzeugung von Zufallszahlen stellt die Klasse `Rechner` zur Verfügung?

Um die Dienste eines Rechners in Anspruch nehmen zu können, muss man vorher ein Objekt `meinRechner` erzeugen. Da der Rechner im Bibliothekspaket `sum.werkzeuge` enthalten ist, müssen Sie zu Beginn des Programms die folgende Zeile ergänzen:

```
import sum.werkzeuge.*;
```

Um einen Stift an eine zufällige Position im Bildschirmfenster zu bewegen, kann man dann folgende Anweisung benutzen:

```
schatzStift.bewegeBis(
    meinRechner.ganzeZufallszahl(0, derBildschirm.breite()),
    meinRechner.ganzeZufallszahl(0, derBildschirm.hoehe()));
```

Es wird zuerst eine ganze Zufallszahl zwischen 0 und der Bildschirmbreite erzeugt. Das ist die neue horizontale Position des Stifts. Das Gleiche passiert mit der vertikalen Position. Sie sehen an dieser Zeile sehr schön, wie man Dienstaufrufe verschachteln kann.

Übung 5.12 Schreiben Sie ein kleines Programm `Himmel`, um an zufälligen Stellen auf dem Bildschirm kleine Sterne zu zeichnen. Das Erzeugen der Sterne wird mit einem Mausdruck beendet.

Übung 5.13 Erweitere das Programm `Himmel` so, dass auch die Farbe der Sterne zufällig ist.

Übung 5.14 Schreibe ein Programm, welches verschieden lange, verschieden farbige Linien an zufälligen Positionen auf den Bildschirm zeichnet. Die Linien sollen nicht verbunden sein und eine zufällige Linienbreite haben.

Übung 5.15 Schreibe ein Programm, welches einen Stift zufällig über den Bildschirm wandern lässt. Die Bewegung des Stifts soll gleichmäßig sein, seine Richtung wird vor jeder Bewegung zufällig geändert.

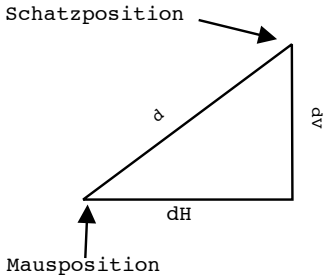
5.10 Schatz verstecken

Als weiteres Projekt soll ein Schatzsuchespiel programmiert werden. Die Grundidee dabei ist die: Der Schatz wird zufällig auf dem Bildschirm versteckt. Der Benutzer klickt auf den Bildschirm und erhält als Information, wie weit vom Schatz er geklickt hat. Wenn man nahe genug am Schatz klickt, wird der Schatz als kleiner gelber Kreis sichtbar. Auch dieses Spiel eignet sich zur mehrstufigen Entwicklung.

Übung 5.16 Erzeugen Sie ein Projekt `Schatz1`. Im Programm soll ein `Schatzstift` erzeugt werden, der an eine zufällige Bildschirmposition bewegt wird. Dort soll ein gefüllter gelber Kreis gezeichnet werden. Wie kann man dafür sorgen, dass der Schatz mindestens 50 Pixel von den Bildschirmrändern entfernt ist?

5.11 Abstand berechnen

Wenn man mit der Maus klickt, soll der Abstand zwischen dem Schatz und der Mausposition berechnet werden. Dazu kann man den Satz des Pythagoras verwenden:



$$d = \sqrt{(MausH - SchatzH)^2 + (MausV - SchatzV)^2}$$

Abbildung 5.2:
Abstand zweier Punkte mit dem Satz des Pythagoras

Die Klasse `Rechner` stellt die benötigten Dienste zur Verfügung. Mit einem weiteren Stift, dem `Schreibstift`, kann das Ergebnis dann auf den Bildschirm geschrieben werden. Ja, es ist möglich, mit mehreren Stiften zu zeichnen. Es macht aber keinen Sinn, mehrere Bildschirme, Mäuse oder Tastaturen zu erzeugen!

```
schreibStift.schreibeZahl(meinRechner.wurzel(
    meinRechner.quadrat(dieMaus.hPosition() -
        schatzStift.hPosition())
    + meinRechner.quadrat(dieMaus.vPosition() -
        schatzStift.vPosition())));
```

Übung 5.17 Bei einem Mausklick soll auf dem Bildschirm oben links der Abstand ausgegeben werden. Das Programm wird mit einem Doppelklick beendet.

Übung 5.18 Der Schatz soll zu Beginn nicht mehr sichtbar sein. Wenn der Abstand kleiner als 5 Pixel ist, soll der Schatz als gelber Kreis sichtbar gemacht werden und das Programm wird beendet.

Eine einfache Möglichkeit zum Löschen der alten Ausgabe besteht darin, ein passendes gefülltes Rechteck zu radieren. Eine Alternative besteht im vorherigen Löschen des Bildschirms.

Bei Übung 5.18 wird eine neue Konstruktion notwendig: Die Schleife wird abgebrochen, wenn ein Doppelklick ausgeführt wurde **oder** wenn der Schatz gefunden wurde. Man spricht hier von einer zusammengesetzten Bedingung. Bedingungen kann man mit **und** und **oder** verknüpfen.

Bei einer **und**-Verknüpfung ist die zusammengesetzte Bedingung nur dann erfüllt, wenn **beide** Teilbedingungen erfüllt sind.

Bei einer **oder**-Verknüpfung ist die zusammengesetzte Bedingung schon dann erfüllt, wenn **nur eine** der Teilbedingungen erfüllt ist.

Für *und* gibt es in Java das Verknüpfungssymbol **&&**, für *oder* das Symbol ||.

Die Durchlaufbedingung für die Suchschleife lautet dann in Pseudocode:

```
solange kein Doppelklick und der Schatz wurde nicht gefunden
```

in Java:

```
while (!dieMaus.doppelKlick() && meinRechner.wurzel(
    meinRechner.quadrat(dieMaus.hPosition()
        - schatzStift.hPosition())
    + meinRechner.quadrat(dieMaus.vPosition()
        - schatzStift.vPosition())) > 5)
```

5.12 Schnell suchen!

Übung 5.19 Sobald der Schatz versteckt wurde, beginnt eine Stoppuhr zu laufen. Wenn eine Minute vergangen ist, soll der nicht gefundene Schatz neu versteckt werden.

Sie benötigen eine Stoppuhr. Die Bibliothek `sum.werkzeuge` stellt eine Uhr zur Verfügung.

Übung 5.20 Rufen Sie im Hilfemenü von BlueJ `Dokumentation sum.werkzeuge` auf und informieren Sie sich über die Klasse `Uhr`. Welche Dienste für eine Stoppuhr stellt die Klasse `Uhr` zur Verfügung?

Übung 5.21 Zeichnen Sie das Struktogramm zur fertigen Schatzsuche

Um die Stoppuhr benutzen zu können, muss natürlich ein Objekt `stoppuhr` der Klasse `Uhr` deklariert und erzeugt werden.

Denkbare Erweiterungen wären ein Punktesystem, abhängig von der Schnelligkeit beim Finden, sowie die Möglichkeit mit der Tastatur zu bestätigen, ob man das Spiel noch mal spielen will.

Falls Sie schon vorher Programmiererfahrung gesammelt haben, werden Sie sich fragen, warum an dieser Stelle noch keine Variablen benutzt wurden. Das Variablenkonzept bei der objektorientierten Programmierung ist etwas anders als bei der imperativen Programmierung und soll erst im nächsten Kapitel eingeführt werden.

5.13 Zusammenfassung

In diesem Kapitel haben Sie die Kontrollstrukturen und deren Darstellung als Struktogramm geübt. Sie haben das Bibliothekspaket `sum.werkzeuge` und die Klassen `Rechner` und `Uhr` kennen gelernt. Sie können jetzt mit Zufallszahlen arbeiten. Sie wissen jetzt, wie man eine Stoppuhr benutzt. Sie haben mit *und* bzw. *oder* verknüpfte Bedingungen kennen gelernt.

Neue Begriffe in diesem Kapitel

• **Animation** Bewegung einer Zeichnung durch wiederholtes Löschen und Neuzeichnen an einer etwas verschobenen Stelle

• **Zufallszahl** Zahl, deren Wert man bei der Erzeugung nicht vorhersagen kann. Zufallszahlen können ganze Zahlen (evtl. in einem bestimmten Bereich, auch negativ!) sein oder Dezimalzahlen zwischen 0 und 1. Zufallszahlen werden mit der Klasse `Rechner` erzeugt.

• **Stoppuhr** Dient zur Zeitmessung. Man kann eine Stoppuhr starten, anhalten und die gestoppte Zeit abfragen. Man kann auch die verflossene Zeit seit Erzeugung der Uhr abfragen.

• **Zusammengesetzte Bedingung** Eine Bedingung kann aus mehreren Teilbedingungen zusammengesetzt sein. Als Verknüpfungen dient der *und*- bzw. *oder*-Operator. Eine mit *und* verknüpfte Bedingung ist nur dann erfüllt (wahr), wenn sämtliche Teilbedingungen wahr sind. Eine mit *oder* verknüpfte Bedingung ist schon dann wahr, wenn eine Teilbedingung wahr ist. Die und-Verknüpfung bindet stärker als die oder-Verknüpfung, *nicht* bindet stärker als *und*. (Ähnliches gilt in der Mathematik: Punktrechnung geht vor Strichrechnung.)

Java-Bezeichner

• **'&&'** ist der *und*-Operator und verknüpft zwei Bedingungen bzw. Wahrheitswerte. && wird als *und* gesprochen.

• '||' ist der *oder*-Operator und verknüpft zwei Bedingungen bzw. Wahrheitswerte. || wird als *oder* gesprochen.

Kapitel 6 Eigene Klassen I

In diesem Kapitel sollen Sie lernen:

- wie man eine Unterklasse der Klasse Stift erzeugt
- wie man Klassen erzeugt, die keine Unterklassen (außer von `Object`) sind
- wie man zu einer eigenen Klasse weitere Unterklassen erzeugt
- wie man in Objekten weitere Objekte erzeugt (hat-Beziehung)
- wie Objekte andere Objekte kennenlernen können (kennt-Beziehung)
- wie man eine Klasse in einem Klassendiagramm darstellt
- wie man die Beziehungen zwischen Klassen in einem Beziehungsdiagramm darstellt
- was man unter den Attributen einer Klasse versteht
- wie man die Attribute und Dienste einer Klasse dokumentiert
- wie man Parameter benutzt und zwischen formalen und aktuellen Parametern unterscheidet.

Bis jetzt haben Sie nur mit vorgegebenen Bibliotheksklassen der SuM-Bibliothek gearbeitet. Sie haben gelernt, wie man in der Klassendokumentation Informationen über die Dienste dieser Klassen erhält. Jetzt sollen Sie eigene Klassen entwickeln und dokumentieren.

Zu Beginn werden Sie lernen, wie man eine vorgegebene Klasse (hier die Klasse `Stift`) erweitert, d.h. dazu Unterklassen bildet. Dies soll dann an einem CAD-Programm geübt werden (CAD = Computer Aided Design). Anschließend wird ein umfangreiches, mehrstufiges Projekt mit dem Namen Billard entwickelt, wobei Sie viele Aspekte der Klassenentwicklung kennen lernen werden.

6.1 Eigene Unterklassen erzeugen

In diesem Abschnitt sollen Sie eine Unterklasse `Figurstift` der Klasse `Stift` erzeugen. Aus der Dokumentation der Klasse `Stift` kennen Sie bereits die Zeichendienste `void zeichneRechteck(double, double)` und `void zeichneKreis(double)` des Stifts. Entsprechend soll im Figurstift ein Dienst `void zeichneQuadrat(double)` ergänzt werden. Die restlichen Dienste erbt der Figurstift vom Stift.

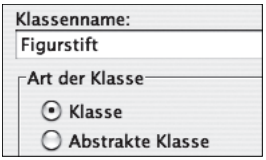


Abbildung 6.1: Erzeugen der Klasse Figurstift

Klicken Sie im Projektfenster in den Knopf `Neue Klasse`, nennen Sie die Klasse `Fi-`

`gurstift` und wählen Sie den Radioknopf `Klasse`. Bestätigen Sie mit `Ok` und ein neues Klassensymbol `Figurstift` erscheint im Projektfenster.

Doppelklicken Sie das Symbol und die Klasse wird im Editor angezeigt. Ergänzen Sie den Text, wie in Abbildung 6.2 angezeigt.

```
1  import sum.kern.*;
2
3  /**
4   * Ein Figurstift ist ein Stift, der auch Quadrate zeichnen kann.
5   *
6   * @author Bernard Schriek
7   * @version 17.7.2005
8   */
9
10 public class Figurstift extends Stift
11 {
12     // Objekte
13
14     // Attribute
15
16     // Konstruktor
17     /**
18      * ein Figurstift wird erzeugt
19      */
20     public Figurstift()
21     {
22         super();
23     }
24
25     // Dienste
26     /**
27      * unabhaengig vom Zustand des Stifts wird ein Quadrat gezeichnet.
28      * @param pGroesse die Seitenlaenge des Quadrats
29      */
30     public void zeichneQuadrat(double pGroesse)
31     {
32         this.zeichneRechteck(pGroesse, pGroesse);
33     }
34 }
```

Abbildung 6.2: Programm der Klasse Figurstift

In Zeile 6 und 7 wurde der Klassenautor und das Datum ergänzt. Sie werden diese beiden Hinweise gleich in der automatisch erzeugten Dokumentation wieder finden..

In Zeile 10 wurde hinter `Figurstift` der Text `extends Stift` ergänzt. Diese beiden Worte bewirken, dass der Figurstift als Unterklasse der Klasse `Stift` erzeugt wird. Der Figurstift erbt also alle Dienste des Stifts.

In Zeile 17 bis 19 wurde ein Kommentar zum Konstruktor ergänzt. Er wird in der automatisch erzeugten Dokumentation wieder auftauchen. Dadurch, dass der Kommentar direkt vor dem Konstruktor steht, wird er dem Konstruktor zugeordnet. Der Konstruktor wird bei der Erzeugung des Objekts ausgeführt.

In Zeile 22 wurde die Anweisung `super();` ergänzt. Damit wird der Konstruktor der Oberklasse (der Klasse `Stift`) aufgerufen.

In Zeile 30 bis 33 wurde der Dienst `zeichneQuadrat` ergänzt. Der Dienst ist von außen

(dem Hauptprogramm) aufrufbar wegen `public`. Der Dienst ist ein Auftrag und keine Anfrage, deshalb `void`. Um das Quadrat zeichnen zu können, wird die Seitenlänge benötigt, die als `double`-Parameter in den Klammern übergeben wird. Das p von `pGroesse` wird immer bei Parametern verwendet, um zu verdeutlichen, dass diese Werte von außen beeinflusst werden.

Das Zeichnen eines Quadrats lässt sich zurückführen auf das Zeichnen eines Rechtecks mit gleicher Länge und Breite, deshalb wird der Dienst `zeichneRechteck` aufgerufen. Diesen Auftrag schickt ein Objekt der Klasse `Figurstift` dann an sich selbst, deshalb das Wort `this` als Empfänger der Nachricht. Da die Klasse `Figurstift` alle Dienste der Klasse `Stift` geerbt hat, versteht der `Figurstift` auch den Auftrag `zeichneRechteck`.

In den Zeilen 26 bis 29 wird der Dienst kommentiert. Die Parameter sollten jeweils hinter dem Schlüsselwort `@param` erklärt werden.

Jetzt sollen Sie die Klassendokumentation erzeugen und ansehen. Dazu wählen Sie im Editor oben rechts `Schnittstelle`.

Abbildung 6.3:
Erzeugen der Dokumentation

Sie sehen jetzt eine Dokumentation ähnlich den schon vorher betrachteten Dokumentationen. Sie sehen die Kommentare und Erläuterungen zu Autor, Version und Parameter.

Diese Dokumentation enthält alle Informationen, die notwendig sind, um mit der Klasse `Figurstift` zu arbeiten. Sie wird in BlueJ als Schnittstelle (zum Benutzer der Klasse) bezeichnet.

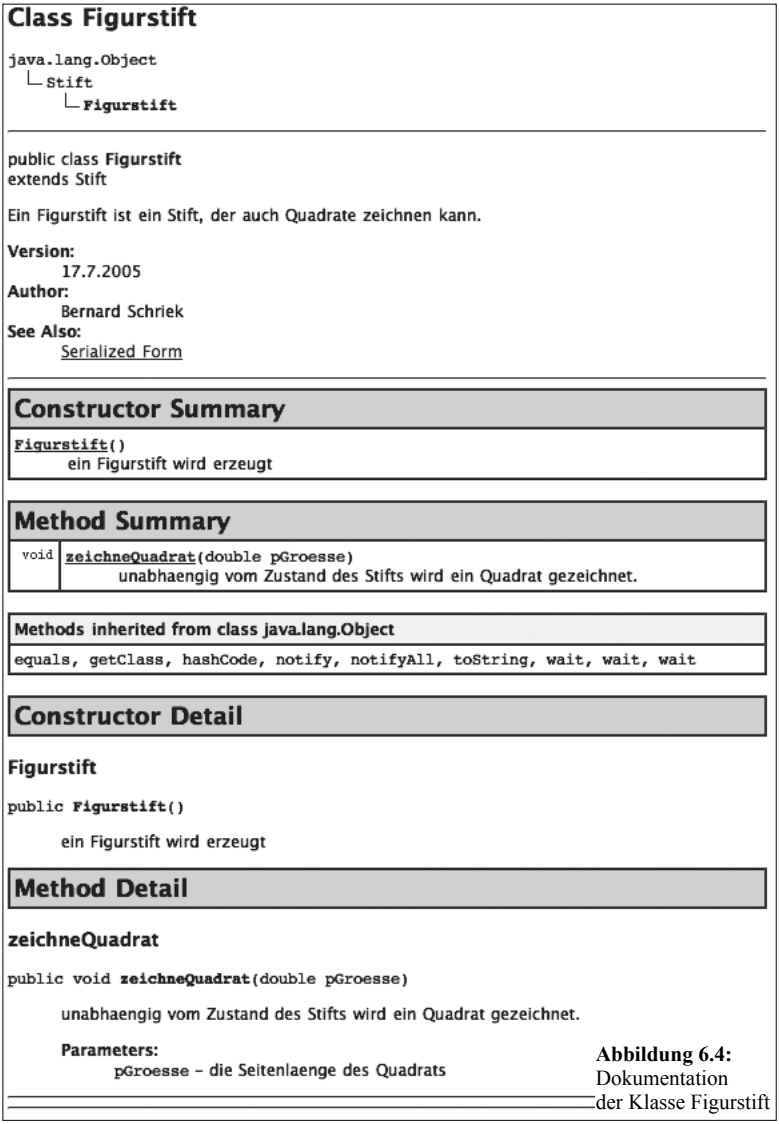
Damit Texte in der Dokumentation erscheinen, müssen Sie in besonderer Form gekennzeichnet werden. Dazu benutzt man die Symbole /** zu Beginn und */ am Ende.

```
/**
 * unabhaengig vom Zustand des Stifts wird ein Quadrat gezeichnet.
 * @param pGroesse die Seitenlaenge des Quadrats
 */
```

Da dieser Text direkt vor dem Dienst `zeichneQuadrat` steht, wird er als Kommentar zu diesem Dienst interpretiert. Es gibt bestimmte Schlüsselworte die mit einem @-Zeichen beginnen. Sie dienen zur Hervorhebung bestimmter Information:

`@author` Autor des Programms (der Klasse).
`@version` Version des Programms
`@param` Erläuterung der Parameter
`@return` Erläuterung des Rückgabewertes bei einer Anfrage

Diese Art der Dokumentation bezeichnet man als *JavaDoc*. Sie ist ein Bestandteil des Java-Pakets.



Class Figurstift

java.lang.Object
  └ Stift
    └ **Figurstift**

---

public class **Figurstift**
extends Stift

Ein Figurstift ist ein Stift, der auch Quadrate zeichnen kann.

Version:
17.7.2005
Author:
Bernard Schriek
See Also:
Serialized Form

Constructor Summary

Figurstift() ein Figurstift wird erzeugt

Method Summary

void	**zeichneQuadrat**(double pGroesse) unabhaengig vom Zustand des Stifts wird ein Quadrat gezeichnet.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Figurstift

public **Figurstift**()

ein Figurstift wird erzeugt

Method Detail

zeichneQuadrat

public void **zeichneQuadrat**(double pGroesse)

unabhaengig vom Zustand des Stifts wird ein Quadrat gezeichnet.

Parameters:
pGroesse – die Seitenlaenge des Quadrats

Abbildung 6.4: Dokumentation der Klasse Figurstift

Diese Seite ist eine HTML-Seite, die vom JavaDoc-Generator erzeugt wurde.

Jetzt soll das Hauptprogramm betrachtet werden, das diese Klasse benutzt. Wechseln Sie von Schnittstelle zu Implementierung zurück und schließen Sie das Fenster. Im Projektfenster erzeugen Sie ein neues SuMKern-Programm und öffnen es im Editor. Ändern Sie wie angezeigt:

```
Stift meinStift;              =>   Figurstift meinStift;
meinStift = new Stift();      =>   meinStift = new Figurstift();
meinStift.schreibeText("Hallo Welt");  =>
                                   meinStift.zeichneQuadrat(80);
```

Übersetzen Sie das Programm und führen Sie es aus. Jetzt sollte ein Quadrat gezeichnet werden.

Übung 6.1 Ergänzen Sie in der Klasse `Figurstift` den Dienst `void zeichneRing(double, double)`, mit dem ein Ring (zwei Kreise) gezeichnet wird. Der erste Parameter soll der innere Radius sein, der zweite Parameter der äußere Radius. Testen Sie den Dienst mit dem Hauptprogramm.

Um den Grundriss eines Hauses interaktiv am Bildschirm zu zeichnen, soll jetzt eine Klasse `Planstift` als Unterklasse der Klasse `Stift` entwickelt werden. Ein Planstift soll das Zeichnen der Grundelemente eines Grundrisses als Dienste zur Verfügung stellen. Es soll folgende Elemente geben: Wand, Fenster, Tür, linke Ecke, rechte Ecke. Die Längen dieser Elemente sind normiert auf 20 * 5 Einheiten. Nur die Ecken haben eine Breite und Höhe von jeweils 5 Einheiten. In Abbildung 6.5 sehen Sie diese Elemente.

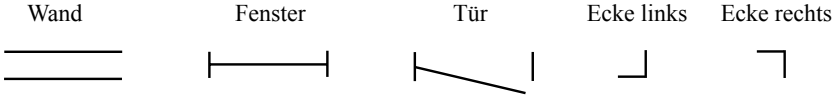


Abbildung 6.5: Grundelemente zum Zeichnen eines Grundrisses

Der Stift soll vor dem Zeichnen in der Mitte der Figur an der linken Seite stehen und nach rechts schauen. Nach dem Zeichnen steht er in der Mitte der Figur an der rechten Seite und schaut nach rechts. Bei den Ecken soll der Stift sich um 90° drehen.

Im Hauptprogramm soll in einer Schleife die Tastatur abgefragt werden. Wenn eine bestimmte Taste gedrückt wird, soll das entsprechende Element gezeichnet werden. Dabei können Zeichnungen entstehen wie in Abbildung 6.6.

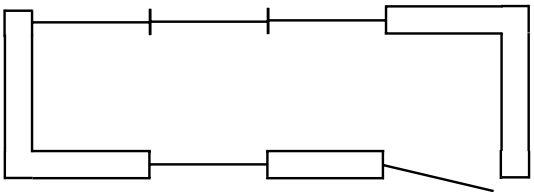


Abbildung 6.6: Grundriss eines Hauses

Programme, mit denen man solche Zeichnungen erstellen kann, bezeichnet man als *CAD-Programme* (CAD = Computer Aided Design = Computer gestützter Entwurf).

Übung 6.2 Entwickeln Sie ein Mini-CAD-Programm, um Grundrisse für Häuser mit der Tastatur zu zeichnen. Benutzen Sie dazu eine Klasse `Planstift`.

Genauso könnte man ein Programm entwickeln, um elektrische Schaltpläne zu zeichnen. Dazu würden Dienste zum Zeichnen der Grundelemente Verbindung, Widerstand, Kon-

densator, Spule, Batterie, Diode usw. benötigt.

Es soll hier nicht verschwiegen werden, dass es der objektorientierten Programmierung besser entsprechen würde, die einzelnen Elemente der Zeichnung als Objekte zu betrachten. Dazu fehlen Ihnen zur Zeit aber noch einige Voraussetzungen. Dies wird in Kapitel 8 nachgeholt werden.

6.2 Klassendiagramm

Jetzt sollen Sie in mehreren Stufen eine Billardsimulation entwickeln. Zu Beginn soll eine Kugel (ein kleiner Kreis) auf dem Bildschirm nach rechts rollen. An dieser Stelle soll objektorientiert geplant werden. Also benötigen Sie ein Objekt der Klasse `Kugel` (und den Bildschirm).

Überlegen Sie zuerst: Welche Dienste sollte eine Kugel zur Verfügung stellen?
Ein Konstruktor gehört zu jeder Klasse. Wir bezeichnen ihn im Klassendiagramm generell mit `init`, denn er ist der Initialisierungsdienst der Klasse. Der Dienst `gibFrei` sollte auch nicht fehlen. Eine Kugel muss sich zeichnen können, also der Dienst `zeichne`. Die Kugel soll sich bewegen, also ein Dienst `bewege`. Weitere Dienste sollen erst später berücksichtigt werden.

Um eine Klasse mit ihren Diensten einfach darzustellen, benutzt man oft Klassendiagramme. Sie sind in der UML (Unified Modelling Language) definiert. Das Klassendiagramm für unsere Kugel sieht so aus:

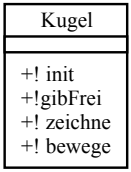


Abbildung 6.7:
Klassendiagramm zur Kugel

Oben steht der Name der Klasse. Nach einem Freiraum folgt eine Auflistung der Dienste. Dabei ist mit `init` immer der Konstruktor gemeint. `gibFrei` ist ein Dienst zum Aufräumen. Das Symbol '+' steht für `public`. Das Ausrufezeichen steht für die Dienstart *Auftrag*. Mit diesen Informationen kann man erst mal einen Prototyp erstellen.

Übung 6.3 Erzeugen Sie ein neues Projekt `Billard1` und erzeugen Sie eine neue Klasse mit Namen `Kugel`. Übertragen Sie den Text aus Abbildung 6.8.

Beachten Sie: Die Kugel hat keine Oberklasse, deshalb fehlt das Schlüsselwort `extends`. *Keine Oberklasse* ist nicht ganz korrekt, denn alle Klassen sind Unterklassen der Klasse `Object`. Falls also in einer Klassendeklaration `extends` fehlt, bedeutet dies `extends Object`.

Bevor der Programmtext geschrieben wird, sollte man im Editor über der Klasse und über den Diensten Dokumentationskommentare ergänzen. Diese JavaDoc-Kommentare beginnen mit '/**' und enden mit '*/'.

Übung 6.4 Ergänzen Sie in der Klasse `Kugel` die Dokumentation der Klasse und der einzelnen Dienste (vgl. Abbildung 6.2). Überprüfen Sie die Dokumentation, indem Sie im Editor von `Implementierung` auf `Schnittstelle` umschalten.

```
1
2  /**
3   * @author Bernard Schriek
4   * @version 18.7.2005
5   */
6  public class Kugel
7  {
8      // Bezugsobjekte
9
10     // Attribute
11
12     // Konstruktor
13     public Kugel()
14     {
15     }
16
17     // Dienste
18     public void gibFrei()
19     {
20     }
21
22     public void zeichne()
23     {
24     }
25
26     public void bewege()
27     {
28     }
29 }
```

Abbildung 6.8:
Prototyp der Klasse Kugel

Jetzt geht es an das Füllen der Dienste mit Programmtext. So etwas nennt man *Implementierung* der Klasse. **Beachten Sie:** Ein Objekt einer anderen Klasse, das Objekten dieser Klasse Aufträge oder Anfragen schickt, muss über die Implementierung nichts wissen. Wichtig ist dafür nur die Schnittstelle, die mit der Dokumentation erzeugt wird. Dies nennt man *Geheimnisprinzip*.

6.3 hat-Beziehung

Da sich die Kugel zeichnen soll, benötigt sie einen Stift. Für das Rahmenprogramm ist dieser Stift uninteressant, das Rahmenprogramm schickt nur Nachrichten an ein Objekt der Klasse `Kugel`. Also erzeugt sich die Kugel selbst einen Stift. Er wird als Bezugsobjekt deklariert:

```
// Bezugsobjekte
private Stift hatStift;
```

Der Name `hatStift` soll anzeigen, dass die Klasse Kugel den Stift **besitzt**. Die Kugel ist also für das Erzeugen und Vernichten des Stifts verantwortlich. Man spricht deshalb von einer **hat-Beziehung**. Bezugsobjekte werden immer als `private` deklariert, denn andere Objekte sollen nicht direkt darauf zugreifen können.

Prinzip: Bezugsobjekte werden immer als `private` deklariert.

Im Konstruktor der Kugel muss der Stift erzeugt werden, im Dienst `gibFrei` der Kugel muss der Auftrag `gibFrei` an den Stift weitergeleitet werden. Beachten Sie, dass im Konstruktor das Wort `void` fehlt. Da der Stift in der Bibliothek `sum.kern` definiert ist, muss als erste Zeile

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

ergänzt werden.

```
// Konstruktor
public Kugel()
{
    hatStift = new Stift();
}


public void zeichne()
{
    hatStift.zeichneKreis(5);
}
```

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


public void bewege()
{
    hatStift.bewegeUm(0.1);
    this.zeichne();
}
```

Übung 6.5 Begründen Sie: Warum ist es sinnvoller `this.zeichne()` zu schreiben als `hatStift.zeichneKreis(5)`?

Übung 6.6 Ergänzen Sie das Hauptprogramm, in dem eine Kugel erzeugt wird. In einer Schleife wird der Kugel solange der Auftrag `bewege()` geschickt, bis die Maustaste gedrückt wird. Testen Sie das Programm. Haben Sie gemerkt, dass im Konstruktor der Kugel noch ein Auftrag an den Stift ergänzt werden muss?

Übung 6.7 Ergänzen Sie einen Dienst `public void loesche()` in der Klasse `Kugel` und rufen Sie diesen Dienst im Dienst `bewege` auf, so dass eine echte Animation wie beim Dartprojekt entsteht.

Im Hauptprogramm sollte der Bildschirm den Bezeichner `hatBildschirm` erhalten, da auch hier eine hat-Beziehung besteht. Das Gleiche gilt auch für die Maus, die jetzt `hatMaus` heißen soll.

6.4 Beziehungsdiagramm

hat-Beziehungen zwischen Klassen bzw. Objekten werden in UML in einem Beziehungsdiagramm dargestellt. Das Hauptprogramm und die Klassen `Bildschirm` und `Maus` sind in Abbildung 6.9 ergänzt.

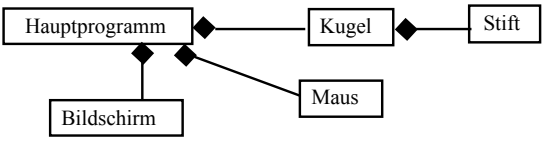


Abbildung 6.9: Beziehungsdiagramm

Das Hauptprogramm hat eine Kugel, die Kugel hat einen Stift. Die gefüllte Raute steht bei der *Besitzerklasse*.

Im Hauptprogramm soll jetzt ein Rahmen (Rechteck) gezeichnet werden, in dem sich die Kugel hin und her bewegen soll. Zum Zeichnen des Rahmens benötigt das Hauptprogramm einen eigenen Stift, der auch `hatStift` heißen kann, da beide Stifte in verschiedenen Klassen deklariert sind. Wenn das Hauptprogramm die Kugel an der Bande abprallen lassen soll, muss es die Kugel nach ihrer Position fragen können (Anfragen `hPosition` und `vPosition`) und mit dem Auftrag `setzeRichtung` die Richtung der Kugel ändern können. Auch der Bildschirm sollte besser `hatBildschirm` heißen.

Übung 6.8 Erweitern Sie die Klasse `Kugel` um die Dienste `hPosition`, `vPosition` und `setzeRichtung`. Erweitern Sie das Hauptprogramm so, dass ein Rahmen gezeichnet wird. Prüfen Sie in der Schleife des Hauptprogramms nach der Bewegung der Kugel, ob sie zu weit nach rechts oder links gerollt ist, und ändern Sie die Richtung der Kugel dann so, dass sie in die entgegengesetzte Richtung rollt.

Die Kugel beantwortet die Anfrage `hPosition()`, indem sie das Ergebnis der Anfrage `hatStift.hPosition()` an Stift zurückgibt.

```
/**
* Die horizontale Position der Kugel wird zurückgegeben
* @return horizontale Position der Kugel
*/
public double hPosition()
{
    return hatStift.hPosition();
}
```

Bei Anfragen wird statt des Schlüsselworts `void` der Ergebnistyp der Anfrage geschrieben. Das ist in diesem Fall double, also eine Dezimalzahl. Jede Anfrage benötigt außerdem eine Programmzeile, die mit dem Schlüsselwort `return` beginnt. Danach wird angegeben, welchen Ergebniswert die Anfrage zurückliefern soll.

Der Auftrag `setzeRichtung` der Kugel ändert den Winkel des Stifts.

```
/**
* die Richtung der Kugel wird geändert
* @param pRichtung neue Richtung der Kugel in Grad
*/
public void setzeRichtung(double pRichtung)
{
    hatStift.dreheBis(pRichtung);
}
```

Informieren Sie sich in durch Wechsel auf `Schnittstelle` über die Wirkung der JavaDoc-Anweisung `@return`.

Das Beziehungsdiagramm hat sich geändert. Jetzt hat auch das Hauptprogramm einen Stift, den es erzeugen und wieder freigeben muss.

Prinzip: Bei einer **hat-Beziehung** ist das besitzende Objekt verantwortlich für die Erzeugung und Freigabe des besessenen Objekts.

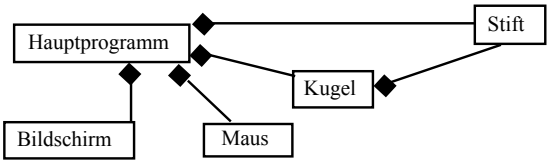


Abbildung 6.10:
Beziehungsdiagramm

Man hätte natürlich das Abprallen von der Klasse Kugel selbst regeln lassen können, statt dies im Hauptprogramm zu regeln. Dann hätte die Überprüfung, ob die Kugel am Rand ist, in der Kugel stattfinden müssen. Was ist besser? Ein Prinzip der objektorientierten Programmierung lautet:

Prinzip: Objekte sollen möglichst viel selbst erledigen.

Damit ist die Antwort klar.

Übung 6.9 Verlegen Sie das Abprallen in die Klasse `Kugel` in den Dienst `bewege`. Schreiben Sie dazu zwei neue Dienste: Die Anfragen `amLinkenRand` und `amRechtenRand`, die zurückgeben, ob die Kugel am Rand ist.

Die Anfrage `amLinkenRand` liefert einen Wahrheitswert zurück. Solche Werte werden in Java als `boolean` bezeichnet. Beim folgenden Programmausschnitt soll der Rahmen den linken Rand an Position 50 haben.

```
/**
* Liefert zurück, ob die Kugel den linken Rand überschritten hat
* @return true, wenn der Rand überschritten wurde, sonst false
*/
private boolean amLinkenRand()
{
    if (this.hPosition() < 50 + 5) // linker Rand + Radius
        return true;
    else
        return false;
}
```

Dies hätte man auch kürzer schreiben können:

```
    return this.hPosition() < 50 + 5;
```

Diese Anfrage ist als `private` deklariert, da sie nur intern von der Klasse benutzt wird. Sie erscheint deshalb auch nicht in der Klassendokumentation, wenn Sie auf `Schnittstelle` wechseln. Private Dienste werden im Klassendiagramm mit einem '-' gekennzeichnet.

6.5 Attribute

Eine Kugel hat verschiedene Eigenschaften, die in Zukunft *Attribute* genannt werden sollen. Zu den Attributen der Kugel gehören: Größe, Position, Richtung, Geschwindigkeit.

Bei unseren bisherigen Kugeln war die Größe durch den Radius beim Zeichnen auf 5 Pixel festgelegt. Die Position wurde im Konstruktor der Kugel mit `hatStift.bewege-`

`Bis(100, 200)` festgelegt. Statt 100, 200 haben Sie vermutlich andere Zahlen gewählt. Die Position der Kugel ändert sich ständig, stimmt aber mit der Position des Stifts überein. Da Attribute nicht doppelt verwaltet werden sollen, lassen wir die Position vom Stift der Kugel verwalten. Das Gleiche gilt für die Richtung. Die Geschwindigkeit kommt im Dienst `bewege` zum Tragen. Dort steht `hatStift.bewegeUm(0.1)`. Falls man diesen Wert vergrößert, wird die Kugel schneller, und umgekehrt.

Prinzip: Attribute sollen nicht mehrfach verwaltet werden um Widersprüche zu vermeiden.

Attribute werden mit dem Buchstaben z gekennzeichnet, z.B. `zGroesse`. Das z kommt vom Wort *Zustandsvariable*, einem anderen Wort für Attribut. Man kann sich das z aber auch leicht merken, wenn man sich vorstellt, das Objekt hat einen Zettel (beginnt auch mit z), auf dem es sich die Werte der Attribute merkt. Attribute sind keine Klassen bzw. Objekte. Sie enthalten Zahlen, Zeichen, Zeichenketten und Wahrheitswerte (`int`, `double`, `char`, `String`, `boolean`). Ein String ist übrigens doch ein Objekt, wie man am großen S schon feststellen kann. Die Klasse `String` spielt aber eine Sonderrolle und wir zählen Strings mit zu den Attributen.

Auf Attribute sollen andere Objekte nicht direkt zugreifen können, um Seiteneffekte zu vermeiden. Java ermöglicht diesen Zugriff, aber es ist ein sehr schlechter Stil, wenn man dies tut. Wenn der Wert von Objekten gelesen oder verändert werden soll, so sollte dies nur über festgelegte Dienste der Klasse geschehen.

Prinzip: Attribute werden immer als `private` deklariert, um Seiteneffekte zu vermeiden. Der Zugriff auf Attribute erfolgt immer über Dienste der Klasse.

Für die Bezeichnung der Dienste zum Lesen bzw. Verändern von Attributen gibt es eine Konvention. Dies soll am Attribut `zGroesse` verdeutlicht werden:

```
public int groesse()
{
    return zGroesse;
}
```

```
public void setzeGroesse(int pGroesse)
{
    zGroesse = pGroesse;
}
```

Beim lesenden Zugriff (Anfrage) auf `zGroesse` wird der Bezeichner des Attributs ohne das z als Dienstbezeichner genommen, beim schreibenden Zugriff (Auftrag) wird das Wort `setze` vor das Attribut geschrieben. Übrigens würde man im Englischen die Worte *getSize* und *setSize* benutzen (size = Größe).

Ein Attribut hat immer drei Eigenschaften, die es charakterisieren:

- Der **Name**, mit dem das Attribut bezeichnet ist. Er beginnt immer mit einem 'z'.
- Der **Typ**, der angibt, was für eine Art von Wert das Attribut enthält (boolean, int, double, String usw.).
- Der **Wert**, der eine konkrete Ausprägung des Typs ist (true, 523, -3.14, "Hallo" usw.).

Man kann sich ein Attribut wie einen Karton vorstellen, der eine Aufschrift, den Namen des Attributs trägt. Die Größe des Kartons entspricht dem Typ, der Inhalt dem Wert des Attributs.

Attribute werden üblicherweise im Konstruktor der Klasse mit einem Anfangswert versehen.

Abbildung 6.11:
Klassendiagramm zur Kugel

Das Klassendiagramm zur Kugel enthält jetzt auch die Attribute.

Übung 6.10 Ergänzen Sie die Attribute `int zGroesse` und `double zGeschwindigkeit` in der Klasse Kugel. Im Konstruktor werden die Werte für `zGroesse` und `zGeschwindigkeit` festgelegt. Ändern Sie die Dienste `bewege` und `zeichne` so ab, dass die Attribute dort benutzt werden. Ergänzen Sie Dienste zum Lesen und Verändern dieser Attribute.

Es ist üblich, beim Erzeugen eines Objekts die Attribute als Parameter zu übergeben. Im Hauptprogramm würde man schreiben:

```
hatKugel1 = new Kugel(80, 100, 5, 0.1);
hatKugel2 = new Kugel(120, 200, 10, 0.15);
```

Es sollen im Hauptprogramm jetzt zwei Kugeln erzeugt werde. Die beiden ersten Parameter sind die horizontale und vertikale Anfangsposition, danach folgt die Größe (Radius), zum Schluss wird die Geschwindigkeit übergeben. In der Schleife im Aktionsteil müssen natürlich beide Kugeln den Auftrag `bewege` erhalten.

Die Klasse Kugel muss angepasst werden.

```
public class Kugel
{
   // Bezugsobjekte
   private Stift hatStift;

   // Attribute
   private int zGroesse;
   private double zGeschwindigkeit;
```

```
    // Konstruktor
    public Kugel(int pH, int pV, int pGroesse,
                 double pGeschwindigkeit)
    {
        hatStift = new Stift();
        hatStift.bewegeBis(pH, pV);
        zGroesse = pGroesse;
        zGeschwindigkeit = pGeschwindigkeit;
    }
...
...
```

Die Attribute zur Position werden an den Stift weitergereicht (`bewegeBis(pH, pV)`).

Bei der Parameterübergabe unterscheidet man zwischen *formalen* und *aktuellen Parametern*. Ein formaler Parameter steht in der Kopfzeile des entsprechenden Dienstes. Er beginnt mit dem Buchstaben 'p'. Davor muss sein Typ angegeben werden. Der aktuelle Parameter wird beim Aufruf des entsprechenden Dienstes angegeben. Hier kann ein konkreter Wert oder aber auch ein Term angegeben werden. Der Wert muss dem Typ des formalen Parameters entsprechen, sonst erzeugt der Compiler einen Übersetzungsfehler. Formale Parameter kann man nur in dem Dienst benutzen, in dem sie deklariert wurden. Um die Werte von Parametern dauerhaft zu verwahren, werden sie üblicherweise in Attributen gespeichert.

Übung 6.11 Ändern Sie das Hauptprogramm so, dass 3 Kugeln benutzt werden.

Übung 6.12 Ändern Sie das Hauptprogramm und die Klasse `Kugel` so ab, dass zu Beginn auch die Richtung der Kugel (0 oder 180) übergeben wird.

Im Beziehungsdiagramm werden die drei Kugeln durch die Zahl 3 am Rahmen der Kugel angegeben. Falls dort keine Zahl steht, ist eine 1 gemeint.

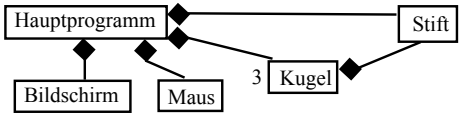


Abbildung 6.12:
Beziehungsdiagramm

Es ist jetzt möglich, im Hauptprogramm beim Erzeugen der Kugeln auch andere Winkel als 0° für rechts und 180° für links anzugeben. Beim Abprallen kommt jetzt noch der obere und untere Rand dazu. Generell gilt für die Winkel:
Einfallswinkel = Ausfallswinkel

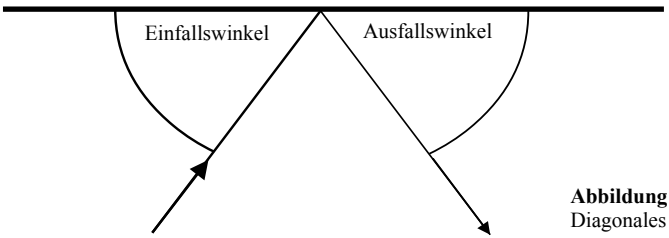


Abbildung 6.13:
Diagonales Abprallen

Um die neue Richtung der Kugel zu berechnen, benötigt man die alte Richtung der Kugel. An Stelle der Richtung der Kugel wird der Winkel des Stiftes genommen.

Falls die Kugel rechts oder links abprallt, berechnet sich die neue Richtung als 180° - alte Richtung. Falls die Kugel oben oder unten abprallt, berechnet sich die neue Richtung als 360° - alte Richtung. Dies kann man mit Hilfe der Winkelgesetze leicht verifizieren.

```
if (this.amLinkenRand() || this.amRechtenRand())
    hatStift.dreheBis(180 - hatStift.winkel());
else if (this.amOberenRand() || this.amUnterenRand())
    hatStift.dreheBis(360 - hatStift.winkel());
```

Damit die Kugel weiß, wann sie oben und unten abprallt, müssen die privaten Anfragen `amOberenRand` und `amUnterenRand` ergänzt werden.

6.6 kennt-Beziehung

In diesem Abschnitt soll das Programm so verändert werden, dass nicht mehr der Rahmen, sondern der Rand des Bildschirmfensters zum Abprallen benutzt wird. Die Klasse `Kugel` kann nicht direkt auf den Bildschirm zugreifen. Der Bildschirm wurde im Hauptprogramm erzeugt und steht nur dort als Bezugsobjekt zur Verfügung.

Sie müssen jetzt dafür sorgen, dass die Kugel den Bildschirm kennen lernt, damit die Kugel dem Bildschirm Nachrichten schicken kann. Dazu ergänzen Sie in der Klasse `Kugel` bei den Bezugsobjekten die Deklaration:

```
private Bildschirm kenntBildschirm;
```

Das Wort `kennt` drückt schon die Art der Beziehung aus. Dieses Objekt wird nicht von der Kugel besessen (hat-Beziehung). Es wird nicht in der Kugel erzeugt und wieder freigegeben. Die einfachste Lösung zum Kennenlernen besteht darin, dass der Bildschirm im Hauptprogramm als weiterer Parameter übergeben wird.

```
hatKugel1 = new Kugel(80, 100, 5, 0.1, 70, hatBildschirm);
```

Der Bildschirm wurde als letzter Parameter übergeben. Der Konstruktor der Kugel wird verändert:

```
// Konstruktor
public Kugel(int pH, int pV, int pGroesse,
             double pGeschwindigkeit, int pRichtung,
             Bildschirm pBildschirm)
{
    hatStift = new Stift();
    hatStift.bewegeBis(pH, pV);
    hatStift.dreheBis(pRichtung);
    zGroesse = pGroesse;
    zGeschwindigkeit = pGeschwindigkeit;
    kenntBildschirm = pBildschirm;
}
```

Eine zweite Möglichkeit wäre ein eigener *Kennenlerndienst* der Kugel:

```
public void lerneKennen(Bildschirm pBildschirm)
```

```
{
    kenntBildschirm = pBildschirm;
}
```

In der Praxis wird meistens die erste Möglichkeit genutzt, denn es gilt in der objektorientierten Programmierung das Prinzip:

Prinzip: Im Konstruktor einer Klasse erhalten alle Bezugsobjekte und Attribute sinnvolle Anfangswerte.

Dies kann natürlich nicht funktionieren, wenn sich zwei Objekte gegenseitig kennen lernen sollen. Dann erhält ein Bezugsobjekt erst im Kennenlerndienst einen sinnvollen Wert.

Übung 6.14 Ändern Sie die Klasse `Kugel` so ab, dass beim Abprallen die Bildschirmränder benutzt werden. Der Rahmen im Hauptprogramm und damit auch der Stift können entfallen.

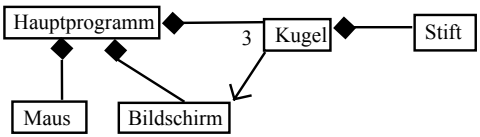


Abbildung 6.14: Beziehungsdiagramm mit kennt-Beziehung

Die kennt-Beziehung wird im Beziehungsdiagramm durch einen Pfeil auf das *gekannte* Objekt dargestellt. Die Kugel kennt den Bildschirm.

Beachten Sie, dass im Konstruktor der Kugel diesmal ein Objekt als Parameter übergeben wurde. In Wirklichkeit wird die Adresse des Speicherbereichs für den Bildschirm übergeben, so dass der Bildschirm jetzt mit zwei Bezeichnern angesprochen werden kann, im Hauptprogramm mit `hatBildschirm` und in der Klasse Kugel mit `kennt-Bildschirm`. Es wird also nur ein Verweis (engl. Reference) auf den Bildschirm übergeben. Man bezeichnet in der Informatik diese Art der Übergabe als *call by reference*.

Jedes Objekt belegt im Hauptspeicher des Computers einen bestimmten Speicherbereich. Beim call by reference wird die Anfangsadresse dieses Speicherbereichs übergeben, so dass der aufgerufene Dienst diesen Speicherbereich dann kennt. Objektbezeichner sind also in Wirklichkeit Namen für Adressen im Hauptspeicher des Computers.

6.7 ist-Beziehung

In diesem Abschnitt sollen Sie Unterklassen der Kugel bilden. Diese Spezialkugeln sollen unterschiedliche Eigenschaften haben. Zwei dieser Unterklassen werden ausführlich besprochen, bei den weiteren Unterklassen ist Ihr Können gefragt.

Die erste Unterklasse der Kugel soll die Reibungskugel sein. Sie wissen, dass eine echte Billardkugel mit der Zeit immer langsamer wird und schließlich stehen bleibt. Dieser Effekt wird durch die Reibung hervorgerufen.

Eine Reibungskugel kann alles, was auch eine normale Kugel kann. Der einzige Unterschied ist, dass sie immer langsamer wird. Die Reibungskugel benötigt also einen zusätzlichen Dienst `bremse`. Dieser Dienst soll immer aufgerufen werden, wenn die Reibungskugel sich bewegt. Es muss also der Dienst `bewege` verändert und der Dienst `bremse` ergänzt werden.

```
private void bremse()
{
    this.setzeGeschwindigkeit(this.geschwindigkeit() * 0.99);
}
```

Die neue Geschwindigkeit soll noch 99% der alten Geschwindigkeit sein. Überlegen Sie sich, warum es nicht sinnvoll ist, regelmäßig einen Wert von der alten Geschwindigkeit abzuziehen.

```
public void bewege()
{
    this.bremse();
    super.bewege();
}
```

Zuerst wird die Geschwindigkeit verringert, danach wird der Dienst `bewege` der Oberklasse, also der Kugel, aufgerufen.

```
// Konstruktor
public Reibungskugel(int pH, int pV, int pGroesse,
   double pGeschwindigkeit, int pRichtung, Bildschirm pBildschirm)
{
    super(pH, pV, pGroesse, pGeschwindigkeit, pRichtung,
          pBildschirm);
}
```

Im Konstruktor wird nur der Konstruktor der Oberklasse aufgerufen und die Parameter werden weiter gereicht. Falls im Konstruktor einer Unterklasse nur der Konstruktor der Oberklasse ohne Parameter (also mit `super();`) aufgerufen wird, kann dieser Konstruktor entfallen. Er wird vom Compiler bei der Übersetzung automatisch ergänzt.

Im Hauptprogramm wird eine vierte Kugel als Reibungskugel deklariert, erzeugt und in der Schleife bewegt.

```
Reibungskugel hatKugel4;
...
hatKugel4 = new Reibungskugel(170, 250, 8, 0.1, 120,
                              hatBildschirm);
...
hatKugel4.bewege();
...
```

Übung 6.15 Erzeugen Sie die Unterklasse `Reibungskugel` der Klasse `Kugel` und testen Sie das Programm.

Als zweite Unterklasse soll eine Farbkugel programmiert werden. Der Farbkugel muss bei der Erzeugung als weiterer Parameter die Farbe übergeben werden.

```
hatKugel5 = new Farbkugel(170, 250, 8, 0.1, 120, hatBildschirm,
                          Farbe.ROT);
```

In der Klasse `Farbkugel` muss ein neues Objekt `Buntstift hatBuntstift` und ein Attribut `int zFarbe` deklariert werden.

```
// Bezugsobjekte
private Buntstift hatBuntstift;

// Attribute
private int zFarbe;
```

Im Konstruktor wird der Buntstift erzeugt und initialisiert. Das Attribut `zFarbe` erhält seinen Wert. Beachten Sie, dass bei `super` der Farbparameter nicht mit übergeben wird.

```
// Konstruktor
public Farbkugel(int pH, int pV, int pGroesse,
                 double pGeschwindigkeit, int pRichtung,
                 Bildschirm pBildschirm, int pFarbe)
{
   super(pH, pV, pGroesse, pGeschwindigkeit, pRichtung,
         pBildschirm);
   hatBuntstift = new Buntstift();
   hatBuntstift.bewegeBis(pH, pV);
   hatBuntstift.setzeFuellmuster(Muster.GEFUELLT);
   hatBuntstift.setzeFarbe(pFarbe);
}
```

Die Dienste `zeichnen` und `loeschen` müssen verändert werden:

```
public void zeichne()
{
    hatBuntstift.bewegeBis(this.hPosition(), this.vPosition());
    hatBuntstift.zeichneKreis(this.groesse());
}

public void loesche()
{
    hatBuntstift.radiere();
    this.zeichnen();
    hatBuntstift.normal();
}
```

So wird erreicht, dass nicht der Stift der Oberklasse sondern der Buntstift der Unterklasse zeichnet. Beim Dienst `zeichne` werden die Attribute der Oberklasse über ihre Zugriffsdienste angesprochen.

Übung 6.16 Erzeugen Sie die Unterklasse `Farbkugel` der Klasse `Kugel` und testen Sie das Programm.

Im Beziehungsdiagramm wird die ist-Beziehung durch einen Pfeil mit einem durchsichtigen Dreieck als Pfeilspitze auf die Oberklasse gekennzeichnet.

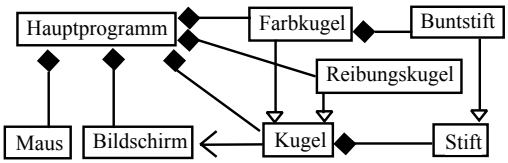


Abbildung 6.15: Beziehungsdiagramm mit ist-Beziehung

Als weitere Spezialkugeln sollen Sie die Pulsierkugel, die Zahlkugel und die Spurkugel programmieren. Hier werden nur ein paar Hinweise gegeben, wie Sie die Aufgaben lösen können, der Schwerpunkt der Arbeit soll aber von Ihnen selbständig erledigt werden.

Die Pulsierkugel soll während des Rollens ihre Größe verändern. Dazu benötigt sie die Attribute `boolean zWaechst, int zMinGroesse, int zMaxGroesse`. Im Konstruktor wird der Anfangswert `zWaechst = true;` festgelegt. Die Minimal- und Maximalgröße soll als Parameter im Konstruktor übergeben werden. Der Dienst `bewegen` muss verändert werden, indem der private Dienst `pulsiere` aufgerufen wird.

```
private void pulsiere()
{
  if (zWaechst)
      this.setzeGroesse(this.groesse() + 1);
  else
      this.setzeGroesse(this.groesse() - 1);
  if (this.groesse() < zMinGroesse || this.groesse() > zMaxGroesse)
      zWaechst = !zWaechst;
}
```

Wenn die Größe der Kugel die Minimalgröße unterschreitet oder die Maximalgröße überschreitet, wird durch Verneinung '!' der Wahrheitswert von `zWaechst` umgedreht, aus `true` wird `false`, aus `false` wird `true`.

Die Zahlkugel hat als Aufschrift eine Zahl. Dazu benötigt sie einen zusätzlichen Stift zum Schreiben der Zahl. Es müssen die Dienste `zeichne` und `loesche` verändert werden. Dabei wird der Stift der Zahlkugel etwas links unten von der Position des Stifts der Kugel gesetzt und die vorher als Parameter übergebene Zahl wird mit dem Dienst `schreibeZahl` geschrieben.

Die Spurkugel hinterlässt beim Bewegen eine Spur (Linie). Dazu benötigt sie einen zusätzlichen Stift. Etwas trickreich ist die Vermeidung des Löschens der Spur durch das Löschen der Kugel. Dazu sollen Sie selbst eine Lösung finden.

Übung 6.17 Erzeugen Sie die Unterklassen `Pulsierkugel`, `Zahlkugel` und `Spurkugel` der Klasse `Kugel` und testen Sie das Programm.

Übung 6.18 Dokumentieren Sie das Projekt und überprüfen Sie die Dokumentation durch Wechsel von `Implementierung` auf `Schnittstelle`.

Übung 6.19 (für leistungsstarke Schüler/innen) Erzeugen Sie eine Schwerkraftkugel als Unterklasse der Reibungskugel, die vom unteren Rand des Bildschirms angezogen wird

Das Projekt `Billard` zeigt hier die Mächtigkeit des Ober- Unterklassen-Konzepts. Die allgemeine Funktionalität (Bewegung, Abprallen) wird von der Oberklasse geregelt. Spezielle Eigenschaften werden in die dann relativ kleinen Unterklassen verlagert.

6.8 Zusammenfassung

In diesem Kapitel haben Sie gelernt, wie man eigene Klassen erzeugt, dokumentiert und benutzt. Dabei wurden viele neue Konzepte und Begriffe eingeführt. In den nächsten Ka-

piteln werden diese Konzepte vertieft und geübt werden. Sie werden beim objektorientierten Entwurf (OOE) regelmäßig Klassendiagramme und Beziehungsdiagramme zeichnen.
• Sie kennen inzwischen drei Arten der Beziehungen. Die ist-Beziehung ist eine Klassenbeziehung, die hat- und kennt-Beziehung ist eine Beziehung zwischen Objekten.
• Sie haben gelernt, dass eine Klasse aus drei Teilen besteht, den Bezugsobjekten, den Attributen und den Diensten.
• Attribute und Dienste werden im Klassendiagramm angegeben, Bezugsobjekte im Beziehungsdiagramm.
•Attribute und Bezugsobjekte sind immer als `private` zu deklarieren. Der Zugriff auf die Attribute erfolgt über standardisierte Dienste.
•Sie haben gelernt, dass eine Klasse für die Erzeugung und Freigabe ihrer hat-Objekte zuständig ist.
•Bei einer kennt-Beziehung wird das kennen zu lernende Objekt normalerweise als Parameter im Konstruktor übergeben.
•Sie haben gelernt, wie man in Unterklassen Dienste verändert (überschreibt) und neue Dienste ergänzt.
•Sie haben gelernt, Klassen mit JavaDoc-Kommentaren zu dokumentieren.
•Sie kennen den Unterschied zwischen formalen und aktuellen Parametern.

Neue Begriffe in diesem Kapitel

• **hat-Beziehung** Wenn ein Objekt ein anderes Objekt erzeugt und verwaltet, so spricht man von einer hat-Beziehung. Solche Bezugsobjekte erkennt man am Wort `hat` vor dem Objektnamen z.B. `hatStift`. Bei einer hat-Beziehung ist die besitzende Klasse für die Erzeugung und Freigabe der besessenen Objekte verantwortlich. Eine hat-Beziehung wird im Beziehungsdiagramm durch eine Verbindungslinie mit einer gefüllten Raute bei der besitzenden Klasse dargestellt.

• **kennt-Beziehung** Wenn ein Objekt ein anderes Objekt im Konstruktor oder einem anderen Dienst kennen lernt, so spricht man von einer kennt-Beziehung. So kann das Objekt auf Dienste des kennt-Objekts zugreifen. Solche Bezugsobjekte erkennt man an dem Wort `kennt` vor dem Objektnamen, z.B. `kenntBildschirm`. Eine kennt-Beziehung wird im Beziehungsdiagramm durch eine Verbindungslinie mit einer Pfeilspitze beim gekannten Objekt dargestellt.

• **Bezugsobjekt** Zu Beginn der Klassendefinition werden die Bezugsobjekte deklariert. Dabei handelt es sich um hat- oder kennt-Beziehungen.

• **Attribut** Eine Eigenschaft eines Objekts bezeichnet man als Attribut. Verschiedene Objekte einer Klasse haben verschiedene Ausprägungen des Attributs. Ein Attribut ist z.B. die Farbe einer Kugel. Attribute sind keine Objekte, sondern primitive Datentypen wie `int` oder `boolean`. Ihre Bezeichner beginnen mit einem z wie `zFarbe`.

• **Zustandsvariable** ist ein anderes Wort für Attribut.

• **Klassendiagramm** Ein Klassendiagramm ist ein Rechteck mit drei Feldern. Oben wird der Name der Klasse eingetragen, danach folgen die Attribute und zum Schluss kommen die Dienste. So erhält man in knapper Form Informationen über die Klasse.

• **Beziehungsdiagramm** Im Beziehungsdiagramm werden die ist-, hat- und kennt-Beziehungen zwischen Klassen durch Symbole gekennzeichnet.

• **Implementierung** Die Klasse wird mit Programmtext gefüllt. Die Implementierung liegt in der Hand des Programmierers, andere Programmierer, die die Klasse verwenden, erhalten nur eine Schnittstellenbeschreibung. Andere Objekte können nur über die Schnittstelle, nicht direkt zugreifen (Geheimnisprinzip).

• **Schnittstelle** Beschreibung einer Klasse sowie der Dienste, die eine Klasse anderen Objekten zur Verfügung stellt. Die Klassen der SuM-Bibliothek und der Java-Bibliothek sind vorübersetzt, d.h. nicht im Quelltext verfügbar. Um sie benutzen zu können, muss ihre Schnittstelle bekannt sein. Dazu dient die Dokumentation, die über das Hilfemenü aufgerufen wird.

• **schreibender Zugriff auf ein Attribut** Dienste, die einen schreibenden (verändernden) Zugriff auf ein privates Attribut einer Klasse ermöglichen, werden üblicherweise mit `setzeAttributsbezeichner` bezeichnet.

• **lesender Zugriff auf ein Attribut** Dienste, die einen lesenden (nicht verändernden) Zugriff auf ein privates Attribut einer Klasse ermöglichen, werden üblicherweise mit dem klein geschriebenen `attributsbezeichner()` bezeichnet. Da Attribute generell als privat gekennzeichnet werden, erfolgt ein Zugriff nur über diese Dienste.

• **formaler Parameter** Ein formaler Parameter wird in der Kopfzeile eines Dienstes in Klammern deklariert. Sein Bezeichner beginnt mit einem 'p' und davor muss sein Typ angegeben werden.

• **aktueller Parameter** Der aktuelle Parameter wird beim Aufruf des Dienstes hinter dem Dienstbezeichner in Klammern angegeben. Er enthält einen konkreten Wert oder einen Term, aus dem der entsprechende Wert berechnet wird.

Java-Bezeichner

• **extends** wird bei der Deklaration einer Unterklasse benutzt. Hinter `extends` steht dann der Name der Oberklasse.

• **boolean** bedeutet Wahrheitswert (`true` oder `false`). Nach George Boole (1815 - 1864)

• **super(...)** steht als erste Anweisung im Konstruktor einer Unterklasse. Damit wird der Konstruktor der Oberklasse aufgerufen.

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