# 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:

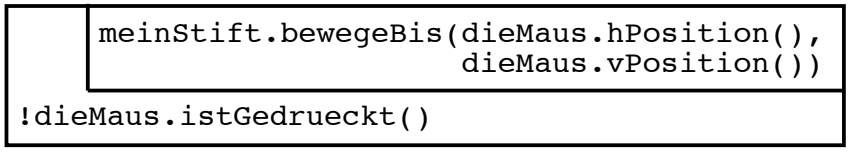


**Abbildung 4.1:** Struktogramm zur Schleife mit Ausgangsbedingung

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:

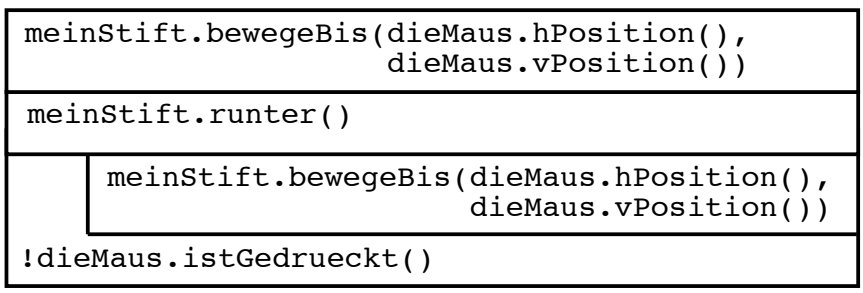


**Abbildung 4.2:** Struktogramm zur Sequenz und Schleife mit Ausgangsbedingung

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:

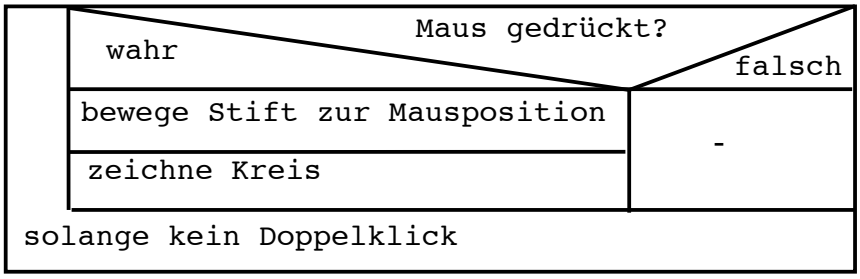


**Abbildung 4.3:** Struktogramm zur einseitigen Verzweigung

**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*:

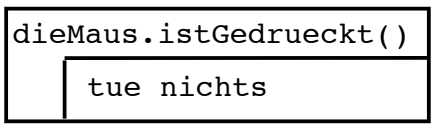


**Abbildung 4.4:** Struktogramm zur Schleife mit Eingangsbedingung

**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;
```

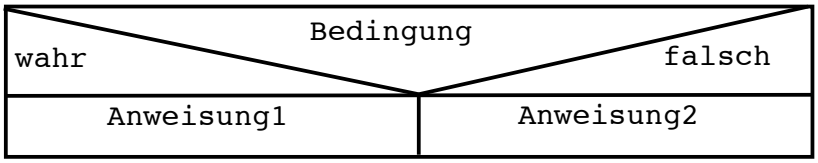


**Abbildung 4.5:** Struktogramm zur zweiseitigen Verzweigung

**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`!

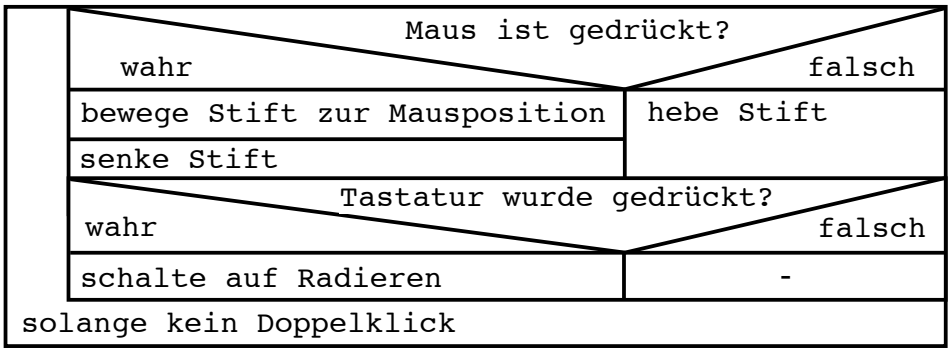


**Abbildung 4.6:** Struktogramm

**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:

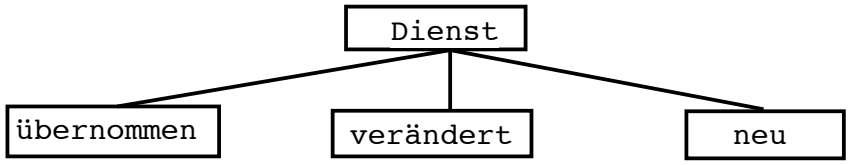


**Abbildung 4.7:** Dienste der Unterklassen

**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.

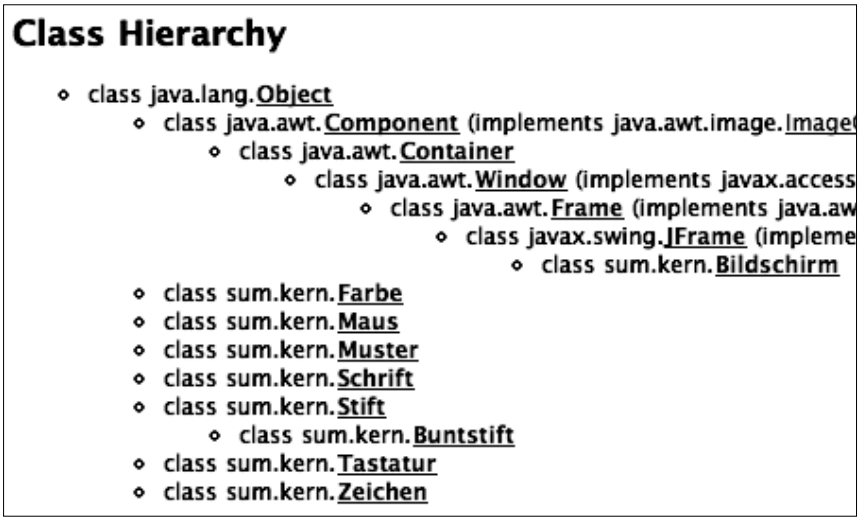


**Abbildung 4.8:** Hierarchie der SuM-Kern-Bibliothek

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.

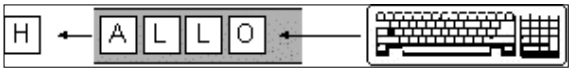


**Abbildung 4.9:**
Tastaturpuffer

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:

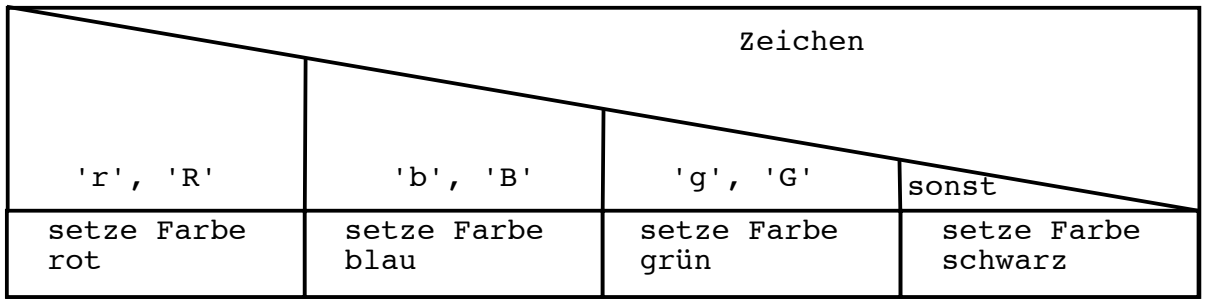


**Abbildung 4.11:**
Struktogramm zur mehrseitigen Verzweigung

**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.