Aufgabensammlung *Grundlagen Java*

Martin Goik[†]

5. Dezember 2000

[†]mailto:goik@HDM-stuttgart.de

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Datumsverarbeitung

Diese Übung dient der Verwendung der Java Klasse `Date`. Die API wird aufgrund der fehlenden Internationalisierung nicht mehr empfohlen, ist aber zu Demonstrationszwecken ausreichend. Als bessere Lösung dient die Klasse `Calendar`.

1.1 Das `Date` API

Unter der URL `http://fb1.HDM-Stuttgart.de/jdk-docs-1.1.6/api/java.util.Date.html` finden Sie die API der Klasse `Date`. Betrachten Sie insbesondere folgende Bestandteile:

- Konstruktor
- Die verschiedenen Zugriffsmethoden
  - getMonth()
  - getDay()
  - getYear()

1.2 Das aktuelle Datum

Nutzen Sie die API zum Ausdruck des aktuellen Datums

1.3 Vergleich zweier Zeiten

Erzeugen Sie via Konstruktor zwei verschiedene `Date` Objekte. Suchen Sie im API nach einer Methode, um festzustellen, welches von beiden Objeken ein jüngeres Datum darstellt.

1.4 Erweiterung

Erweitern Sie das vorherige Beispiel, indem Sie dem Benutzer die Eingaber der beiden Datumszustände ermöglichen.

2 Technische Grundlagen

2.1 Compilation und Start von `Simple.java`

In Analogie zum `Hello, World...` Programm in der Programmiersprache C haben wir:

Listing von **Sources/Hello/Simpel.java**:

```
1: public class Simpel{                          // Klassenname = Dateiname!!
2:   public static void main(String [] args){    // args: Argumentübergabe.
3:     System.out.println("Der erste Schritt..."); // println:Bibliotheksfunktion.
4:   }
5: }
```

Vor der Einführung in die Sprache **Java** betrachten wir die äußere Struktur dieses Programms. Ein doppelter *Slash* „//“ kennzeichnet den Beginn eines Kommentars. Der darauf folgende Text bis zum Zeilenende wird vom Compiler ignoriert und dient der Dokumentation eines Programms.

Das obige Programm enthält eine *Klasse* `Simpel`, welche ihrerseits den Einstiegspunkt des Programms enthält. Aus diesem Grund muß der Dateiname mit *Simpel* beginnen. Da es sich um ein **Java** Programm handelt, lautet der vollständige Name `Simpel.java`.

Um ein **Java** Programm ausführen zu können, muß es zunächst übersetzt (compiliert) werden. Dies erfolgt durch den Compiler `javac`. Dieser erzeugt aus `Simpel.java` den sogenannten Classfile `Simpel.class`.

Der Classfile `Simpel.class` kann nun durch einen **Java**-Interpreter ausgeführt werden. Der vollständige Prozeß sieht daher folgendermaßen aus:

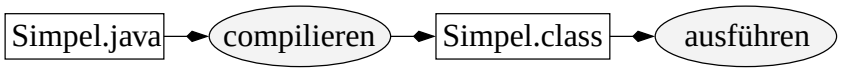


Abbildung 1: Der Bauprozess für `Simple.java`

`Simple.class` besteht aus einer einzigen Klasse `Simple`. Der Einstiegspunkt des Programms *innerhalb* dieser Klasse wird durch das Schlüsselwort `main` definiert. Die Bedeutung des `static` *Modifiers* wird später erklärt. Da das Programm durch `java Simple` gestartet wird, *muß* die Klasse *Simple* eine `main` *Funktion* enthalten. Innerhalb dieser Klasse wird eine Bibliotheksfunktion `println` verwendet, diese dient der Ausgabe eines Textes auf den Bildschirm.

Erstellen Sie dieses Programm mittels einen Editors. Übersetzen Sie das Programm und rufen Sie es anschließend auf.

3 Definition und Konversion von Datentypen

3.1 Implizite und explizite Typumwandlung, Runden

Schreiben Sie ein Programm, welches implizite Typumwandlungen `char` → `int` → `double` und explizite Typumwandlungen (Cast's) `double` → `int` → `char` enthält. Betrachten Sie bei den expliziten Typumwandlungen Zuweisungen mit und ohne Überlauf.

Benutzen Sie die Funktion `Math.round(double)` um die Werte 2.4342 und 2.4352 kaufmännisch auf zwei Nachkommastellen zu runden.

3.2 Überlauf bei `int`

Ein Integer wird in **Java** durch vier Bytes dargestellt. Damit ein solcher Integer positive und negative Zahlen darstellen kann, liegt der Wertebereich im Bereich $\{-2^{n-1}, \ldots, 2^{n-1} - 1\}$. Begründen Sie diese Aussage! Sie können dazu ein vereinfachtes Beispiel mit 3 Bit betrachten.

Schreiben Sie ein Programm, welches die kleinste und größte darstellbare Zahl vom Typ `int` an eine Variable zuweist und ausgibt. Zeigen Sie durch Vermindern bzw. Erhöhen um 1 mit erneuter Ausgabe, daß die Extremaleigenschaft tatsächlich gegeben ist.

4 Operatoren

4.1 Präfix/Postfix Notation von Operatoren

Die Operatoren `++` und `--` gibt es sowohl in Präfix als auch in Infixnotation (z. B. `a++, --a`. Schreiben Sie ein Programm, welches durch Zuweisung an eine zweite Variable den Unterschied demonstriert.

4.2 Rest bei Division

Während `a / b` den ganzzahligen Anteil des Quotienten zweier Integer angibt, bezeichnet `a % b` den „Rest“. Bilden Sie dafür Beispiele. Vergessen Sie nicht die Fälle, in denen `a` und/oder `b` negativ ist/sind.

4.3 Die Operatoren += und Co

Konstruieren Sie jeweils ein Beispiel zur Verwendung der Operatoren `+=, -=, *=, /=` und `%=`.

4.4 Logische Operatoren

Mit den Operatoren `==, !=, <, <=, >, >=` können logische Aussagen gebildet werden. Bilden Sie für jeden Operator ein Beispiel, etwa:

```
System.out.println(“7 < 3:“ + (7 < 3));
```

4.5 Verknüpfung logischer Aussagen

Durch die Operatoren `&&, ||` können Aussagen mit *und* bzw. *oder* verknüpft werden, der Operator `!` dient der Negation. Konstruieren Sie jeweils ein Beispiel.

Zeigen Sie folgende Eigenschaften dieser Verknüpfungsoperatoren für zwei logische Ausdrücke `A` und `B` in **Java**:

- In der *oder*-Verknüpfung `A || B` wird `B` nur dann ausgewertet, falls `A` logisch falsch ist.
- In der *und*-Verknüpfung `A && B` wird `B` nur ausgewertet, falls `A` logisch wahr ist.

Tip: Verwenden Sie zur Demonstration den Inkrementoperator `++`.

5 Strings, Ein-/Ausgabe

5.1 Vergleichsoperationen bei Strings

Java kennt keinen eingebauten Datentyp `String`. Daher liefern die Vergleichsoperatoren `==` und `!=` nicht die Gleichheit/Ungleichheit zweier Zeichenketten, sondern die Aussage, ob zwei Zeichenkettenobjekte identisch sind, oder nicht. Konstruieren Sie zur Demonstration dieses Problems ein Beispiel. Demonstrieren Sie die korrekte Lösung mittels der Objektmethode `equals`.

Hinweis: Es genügt nicht, mittels `String a = “test“, b = “test“;`

den Vergleich `a == b` durchzuführen, weil der **Java** Compiler die Zeichenkette `test` nur einmal als konstantes Objekt anlegt. Erzeugen Sie stattdessen zwei Zeichenketten durch Verkettung kleinerer Zeichenketten.

5.2 Ein-/Ausgabe mittels Corejava

Schreiben Sie ein Demonstrationsprogramm zur Ein/Ausgabe von

- int
- double
- String

Lesen Sie nacheinander den Namen, das Alter und die Körpergröße gemessen in Meter ein. Geben Sie diese Daten *nach allen* Eingaben wieder aus.

Testen Sie Ihr Programm, indem Sie fehlerhafte Eingaben provozieren. Dies ist beispielsweise die Eingabe eines Namens für die Altersangabe.

6 Umwandlung Fahrenheit nach Celsius

6.1 Aufgabe

Die Umwandlung von Temperaturangaben in Grad Fahrenheit (F) nach Grad Celsius (C) erfolgt nach der Formel:

$$C = \frac{5}{9}(F - 32) \tag{1}$$

Schreiben Sie ein Programm, welches den Benutzer zur Eingabe einer Temperatur in Fahrenheit auffordert, und danach die Darstellung in Grad Celsius ausgibt.

7 Telephonkostenabrechnung

7.1 Telephonkostenabrechnung

Es soll folgendes Modell für eine Telephonkostenabrechnung gelten: Pro Monat gibt es 10 Freieinheiten. Jede weitere Einheit wird mit 0.23 Dm berechnet. Die monatliche Grundgebühr für den Anschluß betrage 21.- Dm.

Erstellen Sie ein Programm zur Telephonkostenberechnung. Nach dem Starten des Programms wird der Benutzer zur Eingabe der verbrauchten Einheiten aufgefordert. Anschließend ist folgendes aufzulisten:

- Grundgebuehr
- freie Einheiten
- Preis pro Einheit
- anrechenbare Einheiten
- Endbetrag

Berücksichtigen Sie bitte auch den Fall, daß weniger als zehn Einheiten verbraucht wurden.

7.2 Telephonkostenabrechnung

Erstellen Sie zur Dokumentation des Programms aus 7.1 ein Struktogramm.

8 Zinseszinsen

Es soll ein Algorithmus zur Berechnung von Zinseszinsen erstellt werden. Zur Erinnerung sei das Problem hier kurz skizziert:

Bei jährlicher Verzinsung erhält man für ein gegebenes *Ausgangskapital* K_0 bei einem *Zinssatz* p Zinsen in Höhe von $z = K_0 * \frac{p}{100}$. Das Gesamtkapital K_1 nach einem Jahr beträgt daher:

$$K_1 = K_0 * \left(1 + \frac{p}{100}\right)$$

Wenn man dieses neu erhaltene Kapital weiterverzinst, so erhält man durch Wiederholung die Zinseszinsformel für das Endkapital K_n nach n jähriger Verzinsung:

$$K_n = K_0 * \left(1 + \frac{p}{100}\right)^n \tag{2}$$

Die Berechnung benötigt also folgende Eingabedaten:

- Ausgangskapital K_0
- Zinssatz p
- Verzinsungsdauer n in Jahren

Der Algorithmus soll diese drei Werte einlesen, und zunächst die Gültigkeit von Zinssatz und Laufzeit prüfen, beide müssen positiv sein. Falls einer der zwei Werte unzulässig ist, soll der Algorithmus mit einer qualifizierten Fehlermeldung abbrechen.

Wenn alle drei Werte zulässig sind, soll der Algorithmus zunächst diese Werte ausgeben, und dann eine Tabelle folgender Form erstellen: (Beispiel $p = 5\%$)

Jahr	Zins	Gesamtkapital
0	0	100
1	5	105
...	...	...
n	...	...

8.1 Struktogramm bei einfacher Verzinsung

Formulieren Sie einen Algorithmus als Struktogramm. Die Erzeugung der Tabelle soll dabei als Schleife und ohne Verwendung einer Exponentialfunktion formuliert werden.

8.2 Struktogramm bei Schuldzinsen

Banken berechnen i.A. höhere Schuldzinsen als Kapitalzinsen. Die Verzinsung von Bankschulden erfolgt daher ebenfalls nach der Formel 2, wobei ein höherer Zinssatz eingeht. Formulieren Sie einen zweiten Algorithmus in Form eines Struktogramms, der zwei Zinssätze p_g und p_s einliest, und für $0 \leq K_0$ (Guthaben) den Guthabenzinssatz bzw. für $K_0 < 0$ (Schulden) den Schuldzinssatz benutzt.

8.3 Erstellen eines Java Programms

Realisieren Sie 8.2 als Programm. Nutzen Sie die Methode aus 3.1 zur kaufmännischen Rundung auf zwei Nachkommastellen.

Beachten Sie zunächst den Fall ohne Fehler*behandlung*, indem Sie im Fall fehlerhafter Benutzereingaben mit einer qualifizierten Fehlermeldung abbrechen.

Erweitern Sie das Programm anschließend durch eine Möglichkeit für den Benutzer, fehlerhafte Eingaben zu korrigieren.

9 Sortieren von Intergerwerten

9.1 Sortieren von zwei Integer Werten

Schreiben Sie ein Programm, welches den Benutzer zunächst zur Eingabe zweier Integerwerte auffordert und diese dann aufsteigend nach Größe sortiert+ ausgibt.

9.2 Sortieren von drei Integer Werten

Erweitern Sie 9.1 für Eingabe und Sortierung von drei Werten. Erstellen Sie ein Struktogramm.

10 Ein einfacher Taschenrechner

10.1 Implementierung

Realisieren Sie ein Programm, welches zunächst zwei Gleitkommawerte einliest. Danach soll der Benutzer aufgefordert werden, eine der Operationen (+,-,*,/) durch Angabe einer Zahl auszuwählen, z. B. 1 für die Addition. In Abhängigkeit der eingegebenen Zahlen und der Verlangten Operation ist das Ergebnis zu berechnen.

Hinweis: Verwenden Sie einen Verteiler.

11 Ordinalzahlen in Englisch

Ordinalzahlen werden in englischer Sprache wie folgt dargestellt:

1	1st
2	2nd
3	2rd
4	4th
. . .	. . .
17	17th

11.1 Erzeugung der Endung

Entwerfen Sie ein Programm, das beliebig viele positive, ganze Zahlen einliest und die eingegebene Zahl und die dazugehörige Ordinalzahl in englischer Notation ausgibt. Das Programm soll durch die Eingabe einer Null beendet werden.

11.2

Erweitern Sie das Programm derart, daß für Eingabezahlen kleiner 11 zusätzlich noch die entsprechende römische Zahl ausgegeben wird:

1	1st	I
2	2nd	II
9	9th	IX
17	17th	-

12 Bestimmung von Primzahlen

Eine Primzahl ist eine ganze Zahl größer 1, welche nur durch sich selbst und durch den Wert 1 teilbar ist. Die Folge der Primzahlen lautet daher 2,3,5,7,11,13,. . . .

Der folgende natürlichsprachliche Algorithmus beschreibt, wie entschieden werden kann, ob eine vorgegebene ZAHL eine Primzahl darstellt, oder nicht.

```
Gebe eine ZAHL ein
Setze DIVTEST = 2
SOLANGE (DIVTEST kleiner ZAHL)
   WENN ZAHL durch DIVTEST teilbar
   DANN merke: ZAHL ist keine Primzahl
   erhöhe DIVTEST um 1
WENN gemerkt (ZAHL ist Primzahl)
   DANN Augabe ZAHL ist Primzahl
SONST
   Ausgabe ZAHL ist keine Primzahl
```

12.1 Ein einfacher Algorithmus

Der oben vorgestellte Algorithmus ist nicht sonderlich effizient. Die Schleife wird weiter durchlaufen, auch wenn bereits feststeht, daß ZAHL keine Primzahl ist. Dies wird durch die folgende Verbesserung berücksichtigt:

```
Gebe eine ZAHL ein
Setze DIVTEST = 2
SOLANGE ((DIVTEST kleiner ZAHL) und (ZAHL nicht durch DIVTEST teilbar))
   erhöhe DIVTEST um 1
WENN DIVTEST gleich ZAHL DANN
   Ausgabe ZAHL ist Primzahl
WENN DIVTEST ungleich ZAHL
   DANN Ausgabe ZAHL ist keine Primzahl
```

Formulieren Sie diesen Algorithmus in Form eines Struktogramms.

12.2 Implementierung

Realisieren Sie den in 12.1 vorgestellten Algorithmus in **Java**. Verwenden Sie dazu den aus 4.2 bekannten Operator % zur Restbildung.

12.3 Ein weiter verbesserter Algorithmus

In der Lösung zu 12.2 wird die Schleife zu häufig durchlaufen, wenn es sich um eine Primzahl handelt. Für einen Primzahlkandidaten k reicht es, solange durch kleinere Werte n zu dividieren, bis $\frac{n}{k} < k$ gilt. Dies ist erstmalig der Fall, wenn k gleich dem aufgerundeten Wert von $\sqrt{n}$ ist. Bei der Primzahl 1009 beispielsweise reicht es, bis $\frac{1009}{32}$ zu prüfen.

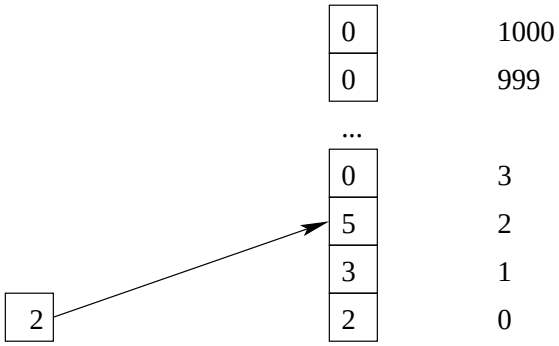
Implementieren Sie diesen verbesserten Algorithmus.

13 Eine Optimierung zur Bestimmung von Primzahlen

13.1 Der verbesserte Algorithmus

In 12 wurden verschiedene Algorithmen dargestellt, um zu entscheiden, ob eine gegebene Zahl eine Primzahl darstellt, oder nicht.

Wir betrachten nun die Aufgabe, die ersten n Primzahlen, z. B. $n = 1000$ zu bestimmen. Erzeugen Sie zu diesem Zweck ein Integerarray der Größe n, in welches Sie die bereits bestimmten Primzahlen eintragen. Zusätzlich benötigen Sie eine Variable, welche den Index der größten bereits in diesem Array abgelegten Primzahl enthält:



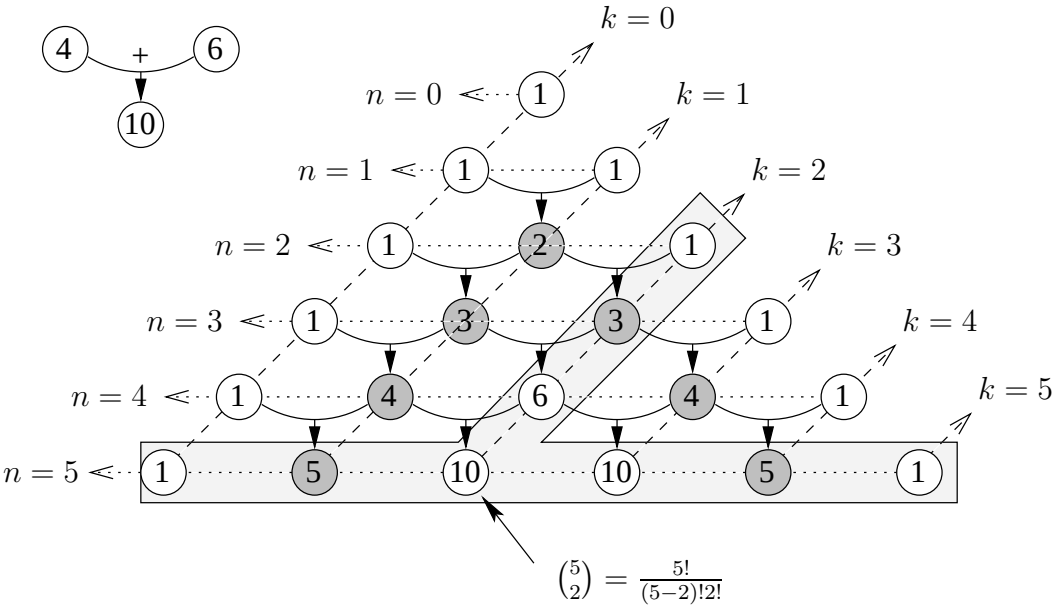
Sie wollen nun entscheiden, ob eine nächsthöhere *ungerade* Zahl, im Beispiel die „7“, eine Primzahl ist, oder nicht. Unter Verwendung der Arrayinhalte reicht es, den Test durch Restbildung auf die bereits ermittelten Primzahlen zu beschränken, welche kleiner sind, als die Wurzel der gegebenen Zahl. Im Beispiel ist dies lediglich die „2“.

14 Lotto „6 aus 49“

Wir betrachten die Lotterie „6 aus 69“. Die Anzahl der möglichen Fälle 6 Kugeln ohne Beachtung der Reihenfolge zu ziehen bedeutet kombinatorisch ein *ungeordnetes Ziehen ohne Zurücklegen*. Die Anzahl der Möglichkeiten, k Kugeln ohne Zurücklegen aus einer Menge von n *unterscheidbaren* Kugeln zu ziehen, beträgt:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)(n-2)\dots(n-k+1)}{k(k-1)(k-2)\dots 3\cdot 2\cdot 1} \tag{3}$$

Diese Koeffizienten $\binom{n}{k}$ heißen Binomialkoeffizienten. Wir betrachten das *PASCAL'sche Dreieck*:



Jeder Eintrag ergibt sich als Summe des linken und rechten oberhalb gelegenen direkten Nachbarn. Im Diagramm dargestellt findet sich der Koeffizient $\binom{5}{2} = 10$. Das Diagramm legt eine rekursive Definition der Binomialkoeffizienten in 3 nahe:

$$\begin{aligned} \text{Induktionsverankerung} &: \text{Für alle} n \in \{0,1,\dots\} : \binom{n}{0} = \binom{n}{n} = 1 \\ \text{Induktionsschritt} &: \binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \end{aligned} \tag{4}$$

Die Äquivalenz beider Definitionen läßt sich per vollständiger Induktion beweisen.

14.1 Dierekte Berechnung

Implementieren Sie eine Klassenmethode zur dierekten Berechnung der Binomialkoeffizienten $\binom{n}{k}$ gemäß der Definition 3 mittels einer Schleife. Verwenden Sie den Datentyp `long`

14.2 Rekursive Berechnung

Implementieren Sie eine zweite Klassenmethode zur rekursiven Berechnung der Binomialkoeffizienten $\binom{n}{k}$ gemäß der Definition 4

14.3 Effektivität

Vergleichen Sie die Laufzeiteffizienz beider Implementierungen anhand der Beispiele $\binom{10}{6}$, $\binom{20}{10}$ und $\binom{20}{15}$. Verwenden Sie zur Zeitmessung die Klassenmethode

```
System.currentTimeMillis()
```

15 Arrays

15.1 Eine mathematische Tafel

Schreiben Sie ein Programm, welches eine Tafel der ersten zehn Quadratzahlen in Form eines Arrays aus `int`'s erstellt. Verwenden Sie dieses Array zur Ausgabe in Form einer Schleife

n	n^2	n	n^2
0	0	5	25
1	1	6	36
2	4	7	49
3	9	8	64
4	16	9	81

15.2 Ausgabe in einer Klassenmethode

Schreiben Sie eine Klassenmethode zur Ausgabe eines Arrays aus Integern. Benutzen Sie diese zur Ausgabe des in 15.1 definierten Arrays.

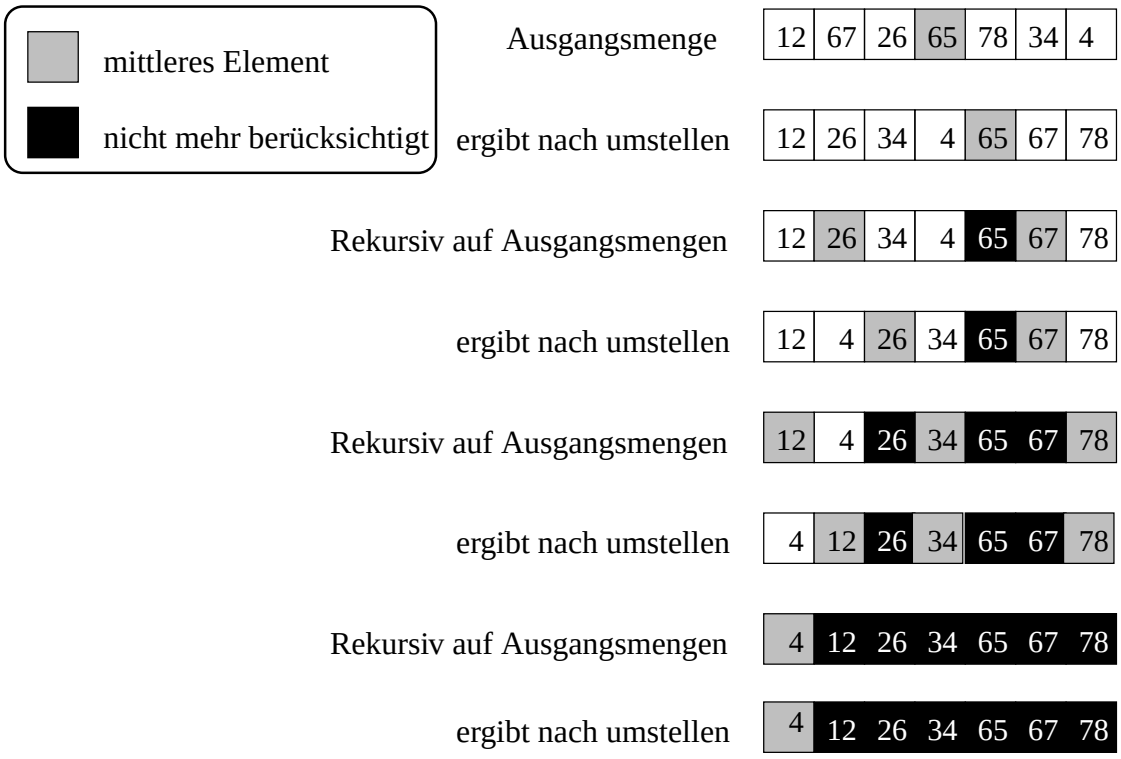
15.3 Arrays und „Call By Reference“

Zeigen Sie, daß ein Array in Java im Gegensatz zu einem eingebauten Datentyp als Referenz übergeben wird.

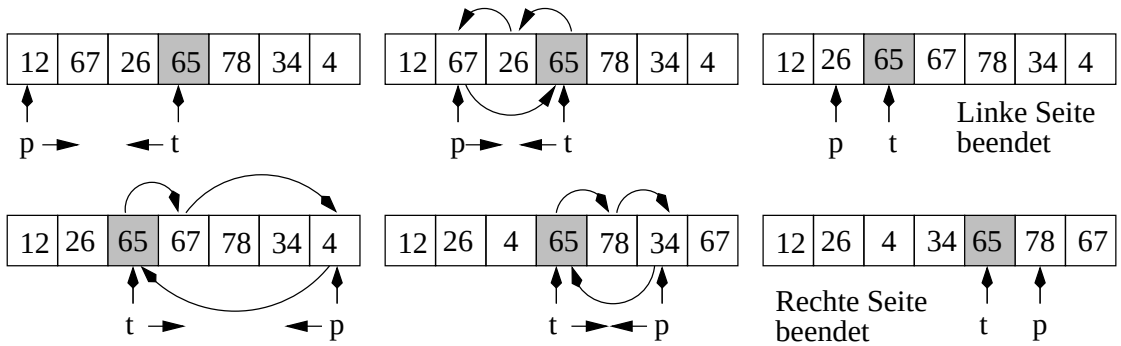
16 Sortieren mittels Quicksort

Quicksort ist ein 1962 von C. A. R. Hoare entwickelter Sortieralgorithmus zur Sortierung eines Arrays von Elementen. Die Elemente dürfen dabei mehrfach vorkommen. Es muß eine Ordnungsrelation definiert sein, welche für jedes Paar (a,b) von Elementen des Arrays festlegt, ob a *kleiner/gleich* b gilt, oder nicht. Der Algorithmus startet mit der Auswahl eines Elementes (Referenzelement) aus dem Vektor gegebener Länge, z.B. dem mittleren. Anschließend werden die anderen Elemente in 2 Mengen aufgeteilt: Die erste Menge enthält Elemente die kleiner sind als das herausgegriffene Referenzelement. Die zweite Menge enthält Elemente, die größer oder gleich dem herausgegriffene Referenzelement sind.

Dieses Verfahren wird dann rekursiv auf beide Untermengen angewendet. Die Rekursion endet bei Mengen mit weniger als 2 Elementen (0 oder 1), da diese nicht weiter sortiert werden müssen. Ein Beispiel:



Die Umstellung innerhalb einer Gruppe kann nach folgendem Schema erfolgen:



16.1 Vertauschen zweier Feldelemente eines Arrays

- Schreiben Sie eine Funktion :

  `int swap(int [] iArray, int i, int j)`, welche in dem Array `iArray` die Werte an den Positionen `i` und `j` vertauscht.

- Schreiben Sie eine Funktion `int getMinIndex(int []iArray)`, welche aus einem Integerarray $(a_0, \ldots, a_n)$ den kleinsten Index eines minimalen Elements bestimmt. Beachten Sie dabei, daß Elemente des Arrays mehrfach vorkommen dürfen.

- Implementieren Sie ein beliebiges Sortierverfahren Ihrer Wahl in Form einer Funktion `void sortSimple(int [] iArray)`. Als Beispiel mag folgender Algorithmus dienen: Wir betrachten ein Array $(a_1, \ldots, a_n)$. Bilden Sie zunächst mittels `getMinIndex()` den Arrayindex k eines kleinsten Arrayelements a_k des Arrays. Vertauschen Sie dann das Element a_k mit dem ersten Element a_0 des Arrays. Sie erhalten so $(a_k, a_1, \ldots, a_0, \ldots, a_n)$.

  Wiederholen Sie dieses Verfahren in einer Schleife, bis das Array vollständig sortiert ist.

16.2 Implementierung von Quicksort

Implementieren Sie das Verfahren für ein `int` Feld fester Länge. Schreiben Sie dazu eine Funktion `void intSort(int [] iArray, int laenge)`, die sich bis zum Erreichen eines Abbruchkriteriums selbst aufruft. Testen Sie `intSort` durch ein `main` Programm, welches den Benutzer auffordert, eine selbst gewählte Anzahl von Zahlen einzulesen. Diese sollen in einem Feld gespeichert und dann sortiert ausgegeben werden.

Hinweis: Es ist hilfreich, eine Funktion

```
cyclicChange (int [] iArray, int a, int b, int c)
```

zu implementieren, welche 3 per index festgelegte Elemente eines Feldes gemäß der Skizze zur Umstellung innerhalb einer Gruppe zyklisch vertauscht.

17 Eine Klasse zur Repräsentation von Brüchen

Ziel dieser Übung ist eine **Java** Implementierung einer Bruch- Klasse. Brüche sind folgende Menge mit Z als Menge der positiven und negativen ganzen Zahlen:

$$\left\{ \frac{z}{n}, z \in Z, n \in Z \backslash \{0\}, z \text{ und } n \text{ teilerfremd} \right\} \tag{5}$$

Die Bedingung „teilerfremd“ bedeutet, daß wir nur Brüche in *maximal gekürzter* Form betrachten.

Für Brüche werden in der üblichen Weise folgende Operationen definiert:

$$\begin{aligned} \text{Addition/Subtraktion} &: \quad \frac{p}{q} \pm \frac{r}{s} = \frac{ps \pm rq}{qs} \\ \text{Multiplikation} &: \quad \frac{p}{q} \cdot \frac{r}{s} = \frac{pr}{qs} \\ \text{Division} &: \quad \frac{\frac{p}{q}}{\frac{r}{s}} = \frac{ps}{qr} \end{aligned} \tag{6}$$

17.1 Attribute und Dezimaldarstellung

Schreiben Sie eine Klasse `Bruch` mit öffentlich sichtbaren Attributen `zaehler` und `nenner`. Addieren Sie eine Methode `double getDezimal()`, um von einem Bruch Objekt den Dezimalwert zu erhalten. Bilden und initialisieren Sie ein Objekt dieser Klasse. Testen Sie die Methode `double getDezimal()`.

17.2 Konstruktor und Ausgabe

Ändern Sie die Bruch Klasse so, daß `zaehler` und `nenner` *private* Attribute der Klasse Bruch werden. Prüfen Sie, daß ein Zugriff z. B. via `objektname.zaehler` nicht mehr möglich ist.

Erweitern Sie die `Bruch` Klasse um folgende Konstruktoren und Zugriffsmethoden für Bruch Objekte:

Konstruktoren	`Bruch()`	Erzeugt den Bruch $\frac{0}{1}$
	`Bruch(int z)`	Erzeugt den Bruch $\frac{z}{1}$
	`Bruch(int z, int n)`	Erzeugt den Bruch $\frac{z}{n}$, falls $n \neq 0$
Zugriffsmethoden	`getZaehler()`	*Lesender* Zugriff auf Zähler
	`getNenner()`	*Lesender* Zugriff auf Nenner
	`ausgabe()`	Ausgabe auf Bildschirm in der Form `z/n`

Testen Sie diese Methoden.

18 Brüche und Operatoren

Ziel dieser Übung ist die Erweiterung der Klasse `Bruch` um Operationen. Jede der untenstehenden Operationen soll mittels `new` einen neuen Bruch als Ergebnis zurückliefern.

18.1 Addition

Implementieren Sie eine Methode `Bruch add(Bruch br)`, welche zwei gegebene Brüche addiert, und einen neuen Bruch zurückliefert.

18.2 2 Multiplikationen

Implementieren Sie unter Verwendung des Overloading Konzeptes zwei Methoden `mult(...)` zur:

1. Multiplikation von einem Bruch mit einer Zahl
2. Multiplikation zweier Brüche miteinander.

18.3 Kehrwert

Implementieren Sie eine Methode zur Kehrwertbildung eines Bruches.

19 Figuren und Vererbung

Als Beispiel zur Vererbung betrachten wir Klassen zur Darstellung elementarer geometrischer Objekte wir Kreis, Quadrat und Rechteck. Nach einer Implementierung ohne Verwendung von Vererbung fassen wir die gemeinsamen Methoden und Attribute in einer Basisklasse `Figur` zusammen.

19.1 Rechteck und Kreis ohne Vererbung

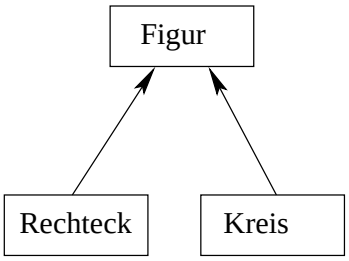
Schreiben eine Klasse `Rechteck` zur Darstellung des geometrischen Objekts Rechteck. Realisieren Sie folgende Methoden:

- Konstruktor `public Rechteck (double x, double y, double k)` zur Erzeugung eines Rechteckes an der Position (x,y) mit Kantenlänge k.
- Methode `public String name()` zur Angabe, um welchen Figurentyp (Rechteck, Kreis,...) es sich handelt.
- Methode `public double getFlaeche()` zur Berechnung der von der Figur eingeschlossenen Fläche.
- Methode `public void skalieren(double faktor)` zur Stauchung/Streckung der Figur um einen Faktor.
- Accessoren `public double getX()` und `public double getY()` für die Mittelpunktskoordinaten.

Implementieren Sie analog dazu eine Klasse für Kreise mit analoger Schnittstelle. Testen Sie beide Klassen durch eine dritte Klasse `Figurtest`

19.2 Rechteck und Kreis mit Vererbung

Verwenden Sie eine abstrakte Basisklasse `Figur` zur Vereinheitlichung gemeinsamer Methoden der Klassen `Rechteck` und `Kreis`. Leiten Sie die Klassen `Rechteck` und `Kreis` von dieser gemeinsamen Basisklasse ab:



19.3 Vererbung und Polymorphie

Demonstrieren Sie auf der Basis von 19.2 den polymorphen Methodenaufruf. Erzeugen Sie dazu eine Referenz auf ein Figurobjekt, z. B. `Figur f= new Kreis(1,2,3)`. Zeigen Sie, daß vermöge `f.xxx()` *dynamisch* die korrekten Methoden wie `name()` aufgerufen werden.

20 Bezug auf Basisklassen mittels `super`

Wir betrachten Erweiterungen zu 19.

20.1 Eigenschaften einer Figur

Schreiben Sie eine Methode `String Eigenschaften()` in Ihrer Klasse Figur. Diese Methode schreibt die Zentrumskoordinaten auf einen String und liefert diesen zurück.

Redefinieren Sie diese Methode in Ihren abgeleiteten Klassen `Rechteck` und `Kreis`. Verwenden sie `super`, um in `Eigenschaften` der abgeleiteten Klasse den Anteil der Basisklasse `Figur` mit einzubinden.

21 Exception Handling

21.1 Der Kehrwert und Ausnahmen

Wir betrachten die Methode der Kehrwertbildung in der Bruchklasse. Im Fall Zaehler gleich null scheitert diese Methode. Werfen Sie in diesem Fall eine `ArithmeticException` Ausnahme aus. Testen Sie diesen Fall durch einen `try ... catch` Block.

21.2 Eine eigene Ausnahmeklasse

Erweitern Sie die Ausnahmebehandlung durch Ableitung einer eigenen Ausnahmeklasse `NennerIstNull` von `ArithmeticException`. Diese Ausnahmeklasse soll per Konstruktor den Nenner des in Frage stehenden `Bruch` Objektes erhalten.

22 Hash-Tabellen

22.1 Key und Value vom Typ `String`

Erzeugen Sie unter Verwendung von `Hashtable` eine Liste von Personen mit Angabe Ihrer jeweiligen Hauptaufgabe:

Key	Value
Schulz	Versandabteilung
Meier	Poststelle und Mahnwesen
Braun	Auftragserfassung und Erstkontakte

Demonstrieren Sie:

- Den Zugriff auf einen bestimmten Mitarbeiter über seinen Key in obiger Tabelle. Schreiben Sie dazu eine Klassenmethode `lookupName(Hashtable h, String s)`, welche einen gegebenen Namen s in h aufsucht und im positiven Fall ausgibt.
- Die Iteration über alle Schlüssel Ihrer Tabelle mit Ausgabe sowohl des Schlüssels selbst, als auch des zugehörigen Wertes. Schreiben Sie dazu eine Klassenmethode `void ausgabeHashtabelle(Hashtable h)`. Sie benötigen dazu den Typ `Enumeration` und die Methode `keys()` aus `Hashtable`

22.2 Key und Value vom Typ `Integer`

Erzeugen Sie eine zu 22.1 analoge Tabelle aus Koordinaten von Parabelwerten (x, x^2) :

Key	Value
-2	4
-1	1
0	0
3	9

Sie können den Typ `int` nicht zum Eintrag in eine `Hashtable` verwenden, sondern müssen auf `Integer` ausweichen.

23 Ein Server für Primzahlen

Wir wollen eine `class Primserver` implementieren. Diese soll der Zerlegung einer beliebigen positiven Ganzzahl in Primfaktoren mit Vielfachheiten dienen. Die Primzahlzerlegung von 90 lautet beispielsweise $90 = 2 \cdot 3 \cdot 3 \cdot 5$. Die Primfaktoren zu 90 lauten also 2,3 und 5 mit den Vielfachheiten 1,2 und 1.

23.1 Der Primserver

Die Klasse `Primserver` soll folgende Schnittstelle haben:

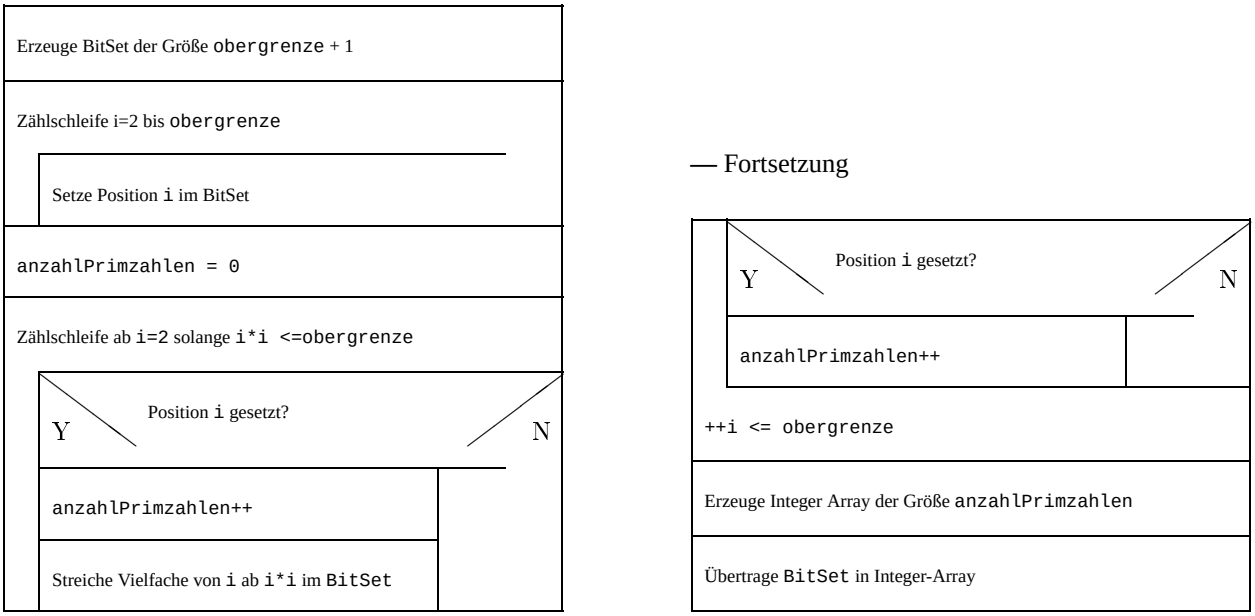
Listing 23.1 *Die Klasse* `Primserver`
Quellcode hier verfügbar

```
import java.util.BitSet;

public class Primserver{                              //Klasse mit genau einer Instanz
  /**/                                                //gemäß Singleton Design Pattern.

  public final static Primserver pserv =              //Es wird genau diese einzige In-
    new Primserver(100);                              //stanz erzeugt, siehe Konstr. .

  private Primserver(int maxPrim)                     //Erzeuge Primz. bis maxPrim.
  throws IllegalStateException{                       //Falls noch keine globale In-
  }
  public final void constructSiebErathostenes         //Bestimme alle Primzahlen bis
    (int obergrenze){                                 //obergrenze.
  }
  public final int getNumPrimes(){                    //Welches ist die Anzahl aller
                                                      //ermittelten Primzahlen?
  }
  public final int getPrimToIndex(int index){         //Zugriff auf Primzahl zu index.

  }
  public final int getGroesstePrimzahl(){             //Welches ist die größte bislang
                                                      //ermittelte Primzahl?.
  }
  private int []                                      //Array der ermittelten Prim-
    primzahlen;                                       //zahlen, primzahlen[0] == 2,
}                                                     //primzahlen[1] == 3, ...   .
```

Ein einfacher Algorithmus zur Bestimmung von Primzahlen bis zu einer maximalen Größe `maxprim` ist das *Sieb des Erathostenes*. Die Implementierung erfolgt am besten in folgenden Schritten:

1. Initialisieren Sie ein `BitSet` für den Indexbereich $[0, \ldots,$ `maxprim`$]$ mit `true` durch Verwendung der Methode `set()`. Dies entspricht der Annahme, daß alle Zahlen aus diesem Intervall Primzahlen sind.
2. Starten Sie ab dem Index 2 mit dem „Herausstreichen“ von Vielfachen des jeweiligen Index aus dem „Sieb“. Diese Operation erfolgt durch Setzen der jeweiligen Position auf `false`, Sie können dazu die Methode `clear()` verwenden. Dies erfolgt solange, bis das Quadrat des Index größer als `maxprim` wird.
3. Das `BitSet` hat jetzt an allen Indizes größer 1 den Wert `true`. Bestimmen Sie die Anzahl aller gefundenen Primzahlen und erzeugen Sie ein `int`-Array dieser Größe. Übertragen Sie die gefundenen Primzahlen aus dem `BitSet` in dieses neue Array. In Ihrer Klasse `Primserv` definieren Sie eine private Variable `primzahlen` als Referenz auf dieses Array, siehe 23.1.

Sieb des Erathostenes — Primzahlbestimmung bis `obergrenze`



Der default- Konstruktor von `Primserv` soll im Gegensatz zu 23.1 zunächst `public` deklariert werden, und alle Primzahlen bis 100 bestimmen.

Testen Sie diese Klasse `Primserv` durch ein Testprogramm, welches die Schnittstelle von `Primserv` zur Ausgabe aller ermittelten Primzahlen verwendet.

23.2 Der Primserver als Singleton

In einer Anwendung wird höchstens ein Primserver benötigt. Aus diesem Grund finden Sie in 23.1 einen als `private` deklarierten Konstruktor mit Angabe einer Obergrenze für die größte zu ermittelnde Primzahl. Zusätzlich finden Sie dort eine Klassenvariable `pserv` vom Typ `final Primserver`. Die Idee dabei ist, daß es nur diesen einen Konstruktor geben soll. Dieser wird bei der Initialisierung von `pserv` aufgerufen und erzeugt ein `Primserver` Objekt. Der Konstruktor prüft, ob `pserv` bereits initialisiert wurde (ungleich `null` ist). Falls dies der Fall war, wird eine Ausnahme vom Typ `IllegalStateException` ausgeworfen.

24 Primfaktorzerlegung

Wir wollen eine `class Primfaktormenge` implementieren. Diese soll der Zerlegung einer beliebigen positiven Ganzzahl in Primfaktoren mit Vielfachheiten repräsentieren. Die Zerlegung wird unter verwendung der Klasse `Primserver` aus 23 durchgeführt. Eine leere Menge von Primfaktoren (also *kein* Primfaktor!) erhält den Wert 1 zugewiesen. Der Wert 0 ist nicht darstellbar.

24.1 Der Konstruktor

Schreiben Sie eine Klasse `Primfaktormenge` mit folgender Schnittstelle:

Listing 24.1 *Die Klasse* `Primfaktormenge`
Quellcode hier verfügbar

```
import java.util.*;

public class Primfaktormenge{
  public Primfaktormenge(int z){
    if (z < 1){
      throw new NoPrimeFactors(z);
    } else {
      primfaktorzerlegung(z);
    }
  }
  public String toString(){                          //Methode zur Ausgabe des Ob-
    String result = new String();                    //jektes als Tabelle aus Prim-
    Enumeration allPrimes = primMulti.keys();        //faktoren mit Vielfachheiten.
    ...
    return result;
  }
  private final void primfaktorzerlegung             //Zerlegung von 2 <= zahl in
    (int zahl){                                      //Primfaktoren mit Vielfachheit.
    ...
  }
  private Hashtable primMulti
    = new Hashtable();
}
```

Da nur Werte größer 0 dargestellt werden können, wirft der Konstruktor im Bedarfsfall eine `NoPrimeFactors` Exception aus.

Die Methode `primfaktorzerlegung` verwendet die vom `Primserver.pserv` zur Verfügung gestellten Primzahlen, um beginnend bei 2 jeweils nach Test auf Restbildung eine Zerlegung in Primfaktoren mit Vielfachheiten zu ermöglichen.

Die Methode `toString` ermöglicht ein bequemes Testen, etwa durch Aufruf von :

```
System.out.println("Zerlegung 147:" + new Primfaktormenge(147));
```

24.2 Weitere Methoden

Erweitern Sie die Klasse `Primfaktormenge` durch folgende Schnittstelle:

Listing 24.2 *Die Klasse* `Primfaktormenge`
Quellcode hier verfügbar

```
import java.util.*;

public class Primfaktormenge{
  //Vorherige Methoden und Attribute
```

```
  public Primfaktormenge ggt                    //Bilde den größten gemeinsamen
    (Primfaktormenge pf){                       //Teiler mit pf.
      //...
    return ggtFaktoren;
  }
  public Primfaktormenge kgv                    //Bilde das kleinste gemeinsame
    (Primfaktormenge pf){                       //Vielfache mit pf.
    //...
    return ...;
  }
  public Primfaktormenge mult                   //Bilde das Produkt mit faktor
    (Primfaktormenge faktor){                   //durch Primfaktoraddition.
    //...
    return ...;
  }
}
```

Verwenden Sie nach Möglichkeit zusätzliche `private` Methoden, um eine möglichst übersichtliche Lösung zu erhalten, denkbar sind folgende Teilprobleme:

- Addieren eines Primfaktors p mit Vielfachheit v zum gegebenen Objekt
- Addieren einer zweiten `Primfaktormenge` zum gegebenen Objekt
- Differenzbildung zweier Primfaktormengen. Dabei müssen die Vielfachheiten berücksichtigt werden.

Testen Sie Ihre Klasse mit folgendem Testprogramm:

Listing 24.3 *Eine Testklasse*
Quellcode hier verfügbar

```
public class Primfaktortest{
  public static void main(String argv[]){
    Primfaktormenge p12 = new Primfaktormenge(12);
    Primfaktormenge p9 = new Primfaktormenge(9);
    System.out.println("Zerlegung 12:\n" + new Primfaktormenge(12));
    System.out.println("Zerlegung 9:\n" + new Primfaktormenge(9));
    System.out.println("Zerlegung 147:\n" + new Primfaktormenge(147));
    System.out.println("GGT(9,12):\n" + p9.ggt(p12));
    System.out.println("KGV(12,9):\n" + p12.kgv(p9));
    }
}
```

1 Datumsverarbeitung

1.1 Das `Date` API

1.2 Das aktuelle Datum

Listing von **Sources/Date/Datetest.java**:

```
 1: import java.util.*;
 2:
 3: public class Datetest{
 4:   public static void main(String [] args){
 5:     Date datum = new Date();
 6:     Date gibtEsNicht = new Date(200,200,200);
 7:     System.out.println(datum.getMonth() + "." +  datum.getDay() + "." +
 8:                        datum.getYear());
 9:     System.out.println(gibtEsNicht.getMonth() + "." +
10:                        gibtEsNicht.getDay() + "." + gibtEsNicht.getYear());
11:   }
12: }
```

1.3 Vergleich zweier Zeiten

Listing von **Sources/Date/Datetest.java**:

```
 1: import java.util.*;
 2:
 3: public class Datetest{
 4:   public static void main(String [] args){
 5:     Date datum = new Date();
 6:     Date gibtEsNicht = new Date(200,200,200);
 7:     System.out.println(datum.getMonth() + "." +  datum.getDay() + "." +
 8:                        datum.getYear());
 9:     System.out.println(gibtEsNicht.getMonth() + "." +
10:                        gibtEsNicht.getDay() + "." + gibtEsNicht.getYear());
11:   }
12: }
```

1.4 Erweiterung

2 Technische Grundlagen

2.1 Compilation und Start von `Simple.java`

Die Lösung wird in der Vorlesung besprochen.

3 Definition und Konversion von Datentypen

3.1 Implizite und explizite Typumwandlung, Runden

Listing von **Sources/Typconv/IntDouble.java**:

```
 1: public class IntDouble{
 2:   public static void main(String [] args){
 3:     char c;
 4:     int i;
 5:     double d;
 6:     c = 'A';
 7:     i = c;
 8:     d = i;                                                    //Implizite Typumw. .
 9:     System.out.println("Konversion char nach int nach double:");
10:     System.out.println(c + " -> " + i + " ->  " + d);
11:     d = -23.12;
12:     i = (int) d;                                              //Explizite Typumw. .
13:     System.out.println("Konversion double nach int:");
14:     System.out.println(d + " -> " +  i);
15:     System.out.println("Kaufmännisches Runden:");
16:     System.out.println(2.4342 + " -> " + Math.round(100 * 2.4342) / 100.);
17:     System.out.println(2.4352 + " -> " + Math.round(100 * 2.4352) / 100.);
18:   }
19: }
```

3.2 Überlauf bei `int`

Die Zweierkomplementdarstellung von n=3 - Bit Integern lautet

-4	1	0	0
-1	1	1	1
-2	1	1	0
-3	1	0	1
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1

Der Wertebereich ist also $\{-2^{n-1}, \ldots, 2^{n-1} - 1\}$.

Listing von **Sources/Typconv/Intueberlauf.java**:

```
 1: public class Intueberlauf{
 2:   public static void main(String [] args){
 3:     int zweiHochDreissig =
 4:       2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 *
 5:       2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 *
 6:       2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2;                          //2^30
 7:     int
 8:       zweitgroessterInt = zweiHochDreissig + (zweiHochDreissig - 2),    //2^31 - 2
 9:       groessterInt = zweitgroessterInt + 1,                             //2^31 - 1
10:       ueberlauf = groessterInt + 1,                                     //2^31
11:       zweitkleinsterInt = - zweiHochDreissig - zweiHochDreissig + 1,    //-(2^31 - 1)
12:       kleinsterInt = zweitkleinsterInt - 1,                             //-2^31
13:       unterlauf = kleinsterInt - 1;                                     //-(2^31 + 1)
14:     System.out.println("zweitgroessterInt:" + zweitgroessterInt);
15:     System.out.println("     groessterInt:" + groessterInt);
16:     System.out.println("        ueberlauf:" + ueberlauf);
17:     System.out.println("zweitkleinsterInt:" + zweitkleinsterInt);
18:     System.out.println("     kleinsterInt:" + kleinsterInt);
19:     System.out.println("        unterlauf:" + unterlauf);
20:   }
21: }
```

Man beachte den Vorzeichenwechsel bei Über/Unterlauf.

4 Operatoren

4.1 Präfix/Postfix Notation von Operatoren

Listing von **Sources/Operator/Simpel.java**:

```
 1: public class Simpel{
 2:   public static void main(String [] args){
 3:     int
 4:       a,
 5:       ausgabe;
 6:     a = 4;
 7:     ausgabe = ++a;                                    // Erhöhen, dann zuweisen.
 8:     System.out.println("ausgabe = ++a:" + ausgabe);
 9:     a = 4;
10:     ausgabe = a++;                                    // Zuweisen, dann erhöhen.
11:     System.out.println("ausgabe = a++:" + ausgabe);
12:     System.out.println("  a:" + ausgabe);
13:   }
14: }
```

4.2 Rest bei Division

Listing von **Sources/Operator/Rest.java**:

```
1: public class Rest{
2:   public static void main(String [] args){
3:    System.out.println("      13 / 4 = " + (13 / 4) + " Rest " + (13 % 4));
4:    System.out.println("   13 / (-4) = " + 13 / (-4) + " Rest " + (13 % (-4)));
5:    System.out.println("   (-13) / 4 = " + ((-13) / 4) + " Rest " + ((-13) % 4));
6:    System.out.println("(-13) / (-4) = " + ((-13) / (-4)) + " Rest " + ((-13) % (-4)));
7:   }
8: }
```

4.3 Die Operatoren += und Co

Listing von **Sources/Operator/Zuweis.java**:

```
 1: public class Zuweis{
 2:   public static void main(String [] args){
 3:     int b = 0;
 4:     b += -2;
 5:     System.out.println("b += -2:" + b);
 6:     b -= 4;
 7:     System.out.println("b -=  4:" + b);
 8:     b *= -5;
 9:     System.out.println("b *= -5:" + b);
10:     b /= 3;
11:     System.out.println("b /=  3:" + b);
12:     b %= 4;
13:     System.out.println("b %=  4:" + b);
14:   }
15: }
```

4.4 Logische Operatoren

Listing von **Sources/Operator/Logisch.java**:

```
1: public class Logisch{
2:   public static void main(String [] args){
3:     System.out.println(" 3 < 7:" + (3 < 7));
4:     System.out.println("4 >= 4:" + (4 >= 4));
5:     System.out.println("4 != 4:" + (4 != 4));
6:     System.out.println(" 4 > 4:" + (4 > 4));
```

```
 7:     System.out.println("4 == 4:" + (4 == 4));
 8:
 9:   }
10: }
```

4.5 Verknüpfung logischer Aussagen

Listing von **Sources/Operator/Verknuepf.java**:

```
 1: public class Verknuepf{
 2:   public static void main(String [] args){
 3:     // Einfache ODER Verknüpfung
 4:     System.out.println("(4 != 4) || (3 < 7):" + ((4 != 4) || (3 < 7)));
 5:     // Einfache UND Verknüpfung
 6:     System.out.println("(4 < 8) && (1 == 2):" + ((4 < 8) && (1 == 2)));
 7:     // Negation
 8:     System.out.println("!(4 < 8):" + (!(4 < 8)));
 9:     int a = 0;
10:     //"Abkürzung" bei ODER Verknüpfung
11:     System.out.println("(4 == 4) ||  (0 < ++a):" + ((4 == 4) ||  (0 < ++a)));
12:     System.out.println("a:" + a);
13:     System.out.println("(4 != 4) ||  (0 < ++a):" + ((4 != 4) ||  (0 < ++a)));
14:     System.out.println("a:" + a);
15:     //"Abkürzung" bei UND Verknüpfung
16:     System.out.println("(4 != 4) &&  (0 < ++a):" + ((4 != 4) &&  (0 < ++a)));
17:     System.out.println("a:" + a);
18:     System.out.println("(4 == 4) &&  (0 < ++a):" + ((4 == 4) &&  (0 < ++a)));
19:     System.out.println("a:" + a);
20:   }
21: }
```

5 Strings, Ein-/Ausgabe

5.1 Vergleichsoperationen bei Strings

Listing von **Sources/String/Stringcompare.java**:

```
 1: public class Stringcompare{
 2:   public static void main(String [] args){
 3:     String
 4:       george = "Big",
 5:       orwell = " Brother",
 6:       summe = george + orwell,          //summe und alles enthalten eine identische
 7:       alles = george + orwell;          //Zeichenkette, sind aber verschiedene Objekte.
 8:     System.out.println(summe + "   ==   " + alles + ":" + (summe == alles));
 9:     System.out.println(summe + " equals " + alles + ":" + (summe.equals(alles)));
10:     System.out.println(summe + " equals " + george + ":" + (summe.equals(george)));
11:   }
12: }
```

5.2 Ein-/Ausgabe mittels Corejava

Listing von **Sources/String/Io.java**:

```
 1: import corejava.*;                           //Enthält die Console Klasse
 2:
 3: public class Io{
 4:   public static void main(String [] args){
 5:     String name =
 6:       Console.readString("Geben Sie bitte Ihren Namen an:");
 7:     int alter =
 8:       Console.readInt("Geben Sie bitte Ihr Alter an:");
 9:     double groesse =
10:       Console.readDouble("Geben Sie bitte Ihre Körpergröße an:");
11:
12:     System.out.println("Die von Ihnen eingegebenen Daten lauten:");
13:     System.out.println("       Name:" + name);
14:     System.out.println("      Alter:" + alter + " Jahre");
15:     System.out.println("Körpergröße:" + groesse + "m");
16:
17:   }
18: }
```

6 Umwandlung Fahrenheit nach Celsius

6.1 Aufgabe

Listing von **Sources/Fahrenheit/Umrechnung.java**:

```
 1: import corejava.*;
 2:
 3: public class Umrechnung{
 4:   public static void main(String [] args){
 5:     double tempInFahrenheit =
 6:       Console.readDouble("Geben Sie bitte eine Temperatur in Fahrenheit ein:");
 7:     System.out.println
 8:       (tempInFahrenheit + " Grad Fahrenheit entspricht " +
 9:        (tempInFahrenheit - 32) * 5 / 12 + "Grad Celsius");
10:   }
11: }
```

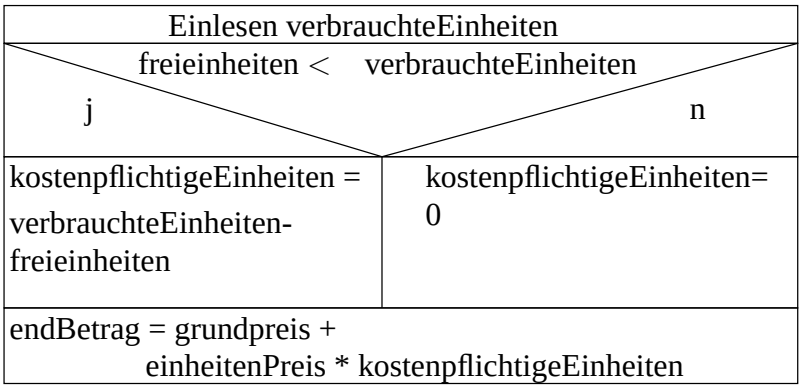
7 Telephonkostenabrechnung

7.1 Telephonkostenabrechnung

Listing von **Sources/Teleabrechnung/Teleabrechnung.java**:

```
 1: import corejava.*;
 2:
 3: public class Teleabrechnung{
 4:
 5:   public static void main(String [] args)
 6:   {
 7:     int freieinheiten = 10;
 8:     double grundPreis = 21.0,
 9:       einheitenPreis = 0.23;
10:
11:     int
12:       kostenpflichtigeEinheiten,
13:       verbrauchteEinheiten =
14:       Console.readInt("Geben Sie die Anzahl verbrauchter Telefon-Einheiten an:\n");
15:
16:     if (freieinheiten < verbrauchteEinheiten){
17:       kostenpflichtigeEinheiten = verbrauchteEinheiten - freieinheiten;
18:     } else {
19:       kostenpflichtigeEinheiten = 0;
20:     }
21:     double endBetrag =
22:       grundPreis + einheitenPreis * kostenpflichtigeEinheiten;
23:
24:     System.out.println("Telefonrechnung:");
25:     System.out.println("Grundgebuehr:" + grundPreis);
26:     System.out.println("anrechenbare Einheiten:" +
27:                        kostenpflichtigeEinheiten);
28:     System.out.println("freie Einheiten:" + freieinheiten);
29:     System.out.println("Preis pro Einheit:" + einheitenPreis);
30:     System.out.println("Endbetrag:" + endBetrag);
31:   }
32: }
```

7.2 Telephonkostenabrechnung

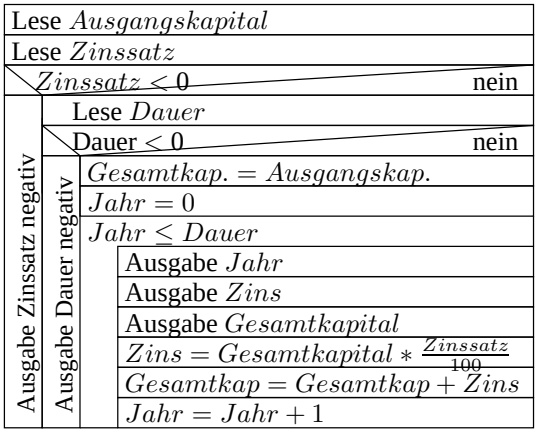


... diverse Ausgaben ...

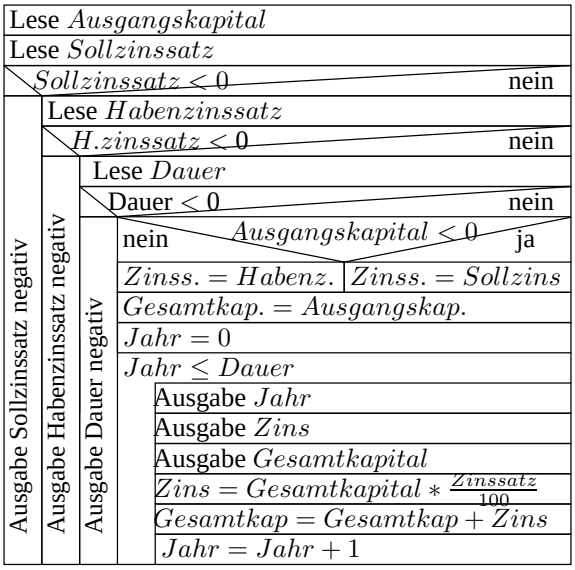
Abbildung 2: Struktogramm zur Telephonkostenabrechnung

8 Zinseszinsen

8.1 Struktogramm bei einfacher Verzinsung



8.2 Struktogramm bei Schuldzinsen



8.3 Erstellen eines Java Programms

Listing von **Sources/Zins/Zins.java**:

```
 1: /*************************************************************
 2: *                                                           *
 3: * Programm zur Berechnung von Zinseszinsen. Als Eingabeparameter*
 4: * dienen                                                    *
 5: * das Ausgangskapital K(0), der Zinssatz z und die Verzinsungs- *
 6: * dauer n. Die Zinseszinsen berechnen sich dann nach der Zin-   *
 7: * seszinsformel:                                             *
 8: *                                                           *
 9: *   K(n) = K(0) * [(1 + (z / 100)) ** n]                    *
10: *  {** = Exponentbildung}                                   *
11: *                                                           *
```

```
12: ************************************************************/
13: import corejava.*;
14:
15: public class Zins{
16:   public static void main(String [] args){
17:
18:     System.out.println("Berechnung von Zinseszinsen\n");
19:
20:     /* Einlesen von StartKap, Zinssatz und Verzinsungsdauer */
21:     System.out.println ("Geben Sie bitte folgende Werte ein:\n");
22:     double startKap = /* Ausgangskapital pos. = haben, negativ = soll */
23:       Console.readDouble ("Startkapital in DM:");
24:     double zinssatz;
25:     if (startKap < 0){
26:       zinssatz =  Console.readDouble("Schuldzinssatz:");
27:     } else {
28:       zinssatz =  Console.readDouble("Guthabenzinssatz:");
29:     }
30:     while (zinssatz  < 0){  /* Korrektur negat. Zinssatz. */
31:       System.out.println("Eingabefehler: Negativer Zinssatz");
32:       System.out.println(" -- Bitte neu eingeben");
33:       System.out.println ("Zinssatz in %: ");
34:       zinssatz = Console.readInt ("Zinssatz in %:");
35:     }
36:     int zinsdauer =
37:       Console.readInt ("Verzinsungsdauer in Jahren: ");
38:     while (zinsdauer  < 0){       /* Korrektur negat. Zinsdauer. */
39:       System.out.println("Eingabefehler: Negative Verzinsungsdauer");
40:       System.out.println(" -- Bitte neu eingeben");
41:       System.out.println("Verzinsungsdauer in Jahren: ");
42:       zinsdauer = Console.readInt ("Verzinsungsdauer in Jahren: ");
43:     }
44:     /* Widerholung der eingelesenen Daten. */
45:     System.out.println("\nEs wurden folgende Werte eingegeben:\n");
46:     System.out.println("Startkapital: " + startKap);
47:     System.out.println("Zinssatz: " + zinssatz + "%");
48:     System.out.println("Verzinsungsdauer:" + zinsdauer + " Jahr" +
49:                        (1 < zinsdauer ? 'e' : ' ')); /* 1 Jahr, 2 Jahre,... */
50:
51:     System.out.println("Jahr Neuzins Gesamtkapital"); /* Tabellenkopf. */
52:
53:     double
54:       endKap = startKap,        //Gesamtkapital.
55:       neuzins = 0;              //Jaehrlicher Zinszuwachs.
56:     for (int i =  0;            //Initialisierung der Laufvariablen.
57:          i <= zinsdauer;        //Abbruchbedingung.
58:          i++){                  //Fortschrittsanweisung.
59:       System.out.println (i + " " +
60:                           (Math.round(100 * neuzins) / 100.) + " " +
61:                           (Math.round(100 * endKap) / 100.));
62:       neuzins = endKap * zinssatz / 100;
63:       endKap += neuzins;    /* analog:  endKap = endKap + neuzins   */
64:     }
65:   }
66: }
```

9 Sortieren von Intergerwerten

9.1 Sortieren von zwei Integer Werten

Listing von **Sources/Intsort/Sortzwei.java**:

```
 1: import corejava.*;
 2:
 3: public class Sortzwei{
 4:
 5:   public static void main(String [] args)
 6:   {
 7:     int
 8:       a = Console.readInt ("Geben Sie bitte eine Ganzzahl ein:"),
 9:       b = Console.readInt ("Geben Sie bitte eine zweite Ganzzahl ein:");
10:     if (a < b){
11:       System.out.println(a + " < " + b);
12:     } else {
13:       System.out.println(b + " < " + a);
14:     }
15:   }
16: }
```

9.2 Sortieren von drei Integer Werten

Listing von **Sources/Intsort/Sortdrei.java**:

```
 1: import corejava.*;
 2:
 3: public class Sortdrei {
 4:   public static void main(String [] args){
 5:     int
 6:       a = Console.readInt ("Geben Sie bitte eine Ganzzahl ein:"),
 7:       b = Console.readInt ("Geben Sie bitte eine zweite Ganzzahl ein:"),
 8:       c = Console.readInt ("Geben Sie bitte eine dritte Ganzzahl ein:");
 9:     if (a <= b && a <= c){
10:       if (b <= c){
11:         System.out.println(a + " <= " + b + " <= " + c);
12:       } else {
13:         System.out.println(a + " <= " + c + " <= " + b);
14:       }
15:     } else if (b <= c && b <= a){
16:       if (a <= c){
17:         System.out.println(b + " <= " + a + " <= " + c);
18:       } else {
19:         System.out.println(b + " <= " + c + " <= " + a);
20:       }
21:     } else {// (c <= a && c <= b)
22:       if (a <= b){
23:         System.out.println(c + " <= " + a + " <= " + b);
24:       } else {
25:         System.out.println(c + " <= " + b + " <= " + a);
26:       }
27:     }
28:   }
29: }
```

<table>
<tr><td colspan="4">Eingabe a</td></tr>
<tr><td colspan="4">Eingabe b</td></tr>
<tr><td colspan="4">Eingabe c</td></tr>
<tr><td colspan="2">ja: $a \leq b$ und $a \leq c$</td><td colspan="2">nein</td></tr>
<tr><td>ja: $b \leq c$</td><td>nein</td><td colspan="2">ja: $b \leq a$ und $b \leq c$</td><td>nein</td></tr>
<tr><td>Ausgabe
$a \leq b \leq c$</td><td>Ausgabe
$a \leq c \leq b$</td><td colspan="2">ja: $a \leq c$</td><td>nein: $a \leq b$</td></tr>
<tr><td></td><td></td><td>Ausgabe
$b \leq a \leq c$</td><td>Ausgabe
$b \leq c \leq a$</td><td>Ausgabe
$c \leq a \leq b$</td><td>Ausgabe
$c \leq b \leq a$</td></tr>
</table>

10 Ein einfacher Taschenrechner

10.1 Implementierung

Listing von **Sources/Rechner/Rechner.java**:

```
 1: import corejava.*;
 2:
 3: public class Rechner{
 4:
 5:   public static void main(String [] args)
 6:   {
 7:     double
 8:       a = Console.readDouble ("Geben Sie bitte eine Zahl ein:"),
 9:       b = Console.readDouble ("Geben Sie bitte eine zweite Zahl ein:");
10:     System.out.println(1 + "='+'");
11:     System.out.println(2 + "='-'");
12:     System.out.println(3 + "='*'");
13:     System.out.println(4 + "='/'");
14:     System.out.println("Anderer Wert = Ende");
15:     int
16:       operation = Console.readInt("Geben Sie bitte eine Operation ein:");
17:     switch(operation){
18:       case 1:
19:         System.out.println("Summe:" + (a + b));
20:         break;
21:       case 2:
22:         System.out.println("Differenz:" + (a - b));
23:         break;
24:       case 3:
25:         System.out.println("Produkt:" + (a * b));
26:         break;
27:       case 4:
28:         if (b != 0){
29:           System.out.println("Quotient:" + (a / b));
30:         } else {
31:           System.out.println("Nenner ist Null!!");
32:         }
33:         break;
34:       default :
35:         break;
36:     }
37:   }
38: }
```

11 Ordinalzahlen in Englisch

11.1 Erzeugung der Endung

Siehe Lösung von 11.2

11.2

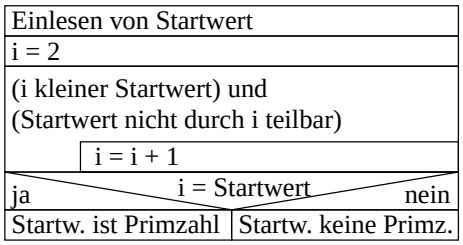
Listing von **Sources/Ordi/Ordi.java**:

```
 1: import corejava.*;
 2:
 3: public class Ordi{
 4:   public static void main(String [] args){
 5:     int eingabe;
 6:     while((eingabe = Console.readInt
 7:            ("Bitte eine Zahl eingeben, 0 = Abbruch")) != 0){
 8:       int entscheid = eingabe % 100;
 9:       if (10 < entscheid && entscheid < 20){
10:         System.out.println(eingabe + "th");
11:       } else {
12:         int lastDigit = eingabe % 10;
13:         switch(lastDigit){
14:         case 1:
15:           System.out.println(eingabe + "st");
16:           break;
17:         case 2:
18:           System.out.println(eingabe + "nd");
19:           break;
20:         case 3:
21:           System.out.println(eingabe + "rd");
22:           break;
23:         case 0:
24:         case 5:
25:         case 6:
26:         case 7:
27:         case 8:
28:         case 9:
29:           System.out.println(eingabe + "th");
30:           break;
31:         default:System.out.println("Sprung hierher unmöglich");
32:         }
33:         if (eingabe < 11){
34:           switch(eingabe){
35:           case 1:
36:             System.out.println("I");
37:             break;
38:           case 2:
39:             System.out.println("II");
40:             break;
41:           case 3:
42:             System.out.println("III");
43:             break;
44:           case 4:
45:             System.out.println("IV");
46:             break;
47:           case 5:
48:             System.out.println("V");
49:             break;
50:           case 6:
51:             System.out.println("VI");
52:             break;
53:           case 7:
54:             System.out.println("VII");
55:             break;
56:           case 8:
57:             System.out.println("VIII");
58:             break;
59:           case 9:
60:             System.out.println("IX");
61:             break;
62:           case 10:
63:             System.out.println("X");
```

```
64:             break;
65:           }
66:         }
67:       }
68:     }
69:   }
70: }
```

12 Bestimmung von Primzahlen

12.1 Ein einfacher Algorithmus



12.2 Implementierung

Listing von **Sources/Prim/Prim.java**:

```
 1: //Programm zur Bestimmung von Primzahlen. Nach Eingabe eines Wertes durch den
 2: //Anwender wird in einer Schleife ermittelt, ob es sich dabei um eine Primzahl
 3: //handelt, oder nicht. Dieses Ergebnis wird ausgegeben.
 4:
 5: import corejava.*;
 6:
 7: public class Prim{
 8:   public static void main(String [] args){
 9:     int primzahlKandidat = Console.readInt ("Zu prüfende Zahl:");
10:     int i;
11:     for (i = 2;                                  //Initialisierung Schleifenvariable.
12:          i < primzahlKandidat &&                 //Abbruchbed.
13:            primzahlKandidat % i != 0;            //Teiler gefunden?
14:          i++);                                   //Fortschrittsanweisung.
15:     if (i == primzahlKandidat){                  //Wurde Teiler gefunden?.
16:       System.out.println(primzahlKandidat +
17:                          " ist eine Primzahl");
18:     } else {
19:       System.out.println(primzahlKandidat +
20:                          " ist keine Primzahl");
21:       System.out.println
22:         ("Ein möglicher Teiler ist " + i);
23:     }
24:   }
25: }
```

12.3 Ein weiter verbesserter Algorithmus

Listing von **Sources/Prim/Primopt.java**:

```
 1: //Verbesserte Version zur Bestimmung von Primzahlen. Die Schleife wird jetzt
 2: //nur noch solange durchlaufen, bis die Wurzel des Primzahlkandidaten
 3: //erreicht ist.
 4:
 5: import corejava.*;
 6:
 7: public class Primopt{
 8:   public static void main(String [] args){
 9:     int primzahlKandidat =
10:       Console.readInt ("Zu prüfende Zahl:");
11:     boolean
12:       istPrimzahl = (1 < primzahlKandidat);     //Annahme, es ist eine Primzahl.
13:     for (int i = 2;                             //Initialisierung Schleifenvariable.
14:          istPrimzahl &&                         //Annahme noch richtig?
15:            i < primzahlKandidat / i;            //Bis hierher reicht es zu prüfen.
16:          i++){                                  //Fortschrittsanweisung.
17:       if (primzahlKandidat % i == 0){
18:         istPrimzahl = false;                    //Annahme war falsch.
19:       }
```

```
20:       if (primzahlKandidat / i <= i){             //Wurde Teiler gefunden?.
21:         System.out.println(primzahlKandidat +
22:                            " ist eine Primzahl");
23:       } else {
24:         System.out.println(primzahlKandidat +
25:                            " ist keine Primzahl");
26:         System.out.println
27:           (i + " ist ein möglicher Teiler ");
28:       }
29:     }
30:   }
31: }
```

13 Eine Optimierung zur Bestimmung von Primzahlen

13.1 Der verbesserte Algorithmus

Listing von **Sources/Prim/Primarray.java**:

```
 1: //Programm zur Bestimmung von Primzahlen. Nach Eingabe eines Wertes durch den
 2: import corejava.*;
 3:
 4: public class Primarray{
 5:   public static void main(String [] args){
 6:     int anzahlPrimzahlen = Console.readInt
 7:       ("Wie viele Primzahlen wollen Sie bestimmen?");
 8:     int [] primArray = bestimmePrimzahlen(anzahlPrimzahlen);
 9:     arrayAusgabe(primArray);
10:   }
11:   public static void arrayAusgabe(int [] arrayRef){
12:     for (int i = 0; i < arrayRef.length; i++){
13:       System.out.println(i + ":" + arrayRef[i]);
14:     }
15:   }
16:   public static int [] bestimmePrimzahlen(int anzahl){
17:     int [] primArray = new int [anzahl];               //Platz für anzahl Primzahlen.
18:     int                                                //Index der zuletzt ermitelten
19:       maxindex = 0,                                    //Primzahl.
20:       maxprim,                                         //Größte, aktuelle Primzahl.
21:       i,                                               //Laufvariable.
22:       j;                                               //Laufvariable.
23:     primArray[maxindex] = 2;
24:     int wurzel;
25:     boolean istPrimzahl;
26:     maxprim = 3;
27:     for (i = 1; i < anzahl; i++){                      //Aktueller Arrayindex
28:       do{
29:         istPrimzahl = true;                            //Annahme, maxprim ist Pz.
30:         wurzel = (int) Math.sqrt(maxprim);             //Bis zu dieser Pz. testen.
31:         for (j = 0;
32:              j < i && primArray[j] <= wurzel;
33:              j++){
34:           if (maxprim % primArray[j] == 0){
35:             istPrimzahl = false;                       //War wohl nix.
36:             break;
37:           }
38:         }
39:         if (istPrimzahl){
40:           primArray [i] = maxprim;                     //Primzahl eintragen
41:         }
42:         maxprim += 2;
43:       } while (!istPrimzahl);                          //Weiter, bis Pz. gefunden.
44:     }
45:     return primArray;
46:   }
47: }
```

14 Lotto „6 aus 49“

14.1 Dierekte Berechnung

Siehe Lösung von 14.2

14.2 Rekursive Berechnung

Listing von **Sources/Binom/Binom.java**:

```
 1: import corejava.*;
 2:
 3: public class Binom{
 4:   public static void main(String [] args){
 5:     int
 6:       anzahlLottokugel = Console.readInt
 7:       ("Wieviele Kugeln gibt es?"),
 8:       anzahlGezogenerKugeln = Console.readInt
 9:       ("Wieviele Kugeln haben Sie gezogen?");
10:     long start = System.currentTimeMillis();
11:     System.out.println                          //Ausgabe Schleifenberechnung.
12:       ("Schleife:" +
13:        binomSchleife(anzahlLottokugel,
14:                      anzahlGezogenerKugeln));
15:     long dauerSchleife = System.currentTimeMillis() - start;
16:     System.out.println("Dauer:" + dauerSchleife);
17:     start = System.currentTimeMillis();
18:     System.out.println                          //Ausgabe rekursive Berechnung.
19:       ("Rekursiv:" +
20:        binomRekursiv(anzahlLottokugel,
21:                      anzahlGezogenerKugeln));
22:     long dauerRekursiv = System.currentTimeMillis() - start;
23:     System.out.println("Dauer:" + dauerRekursiv);
24:     System.out.println("Faktor:" + dauerRekursiv / (double) dauerSchleife);
25:   }
26:   public static long binomSchleife(long n, long k){
27:     long
28:       i,                                        //Laufvariable
29:       ergebnis = n - k + 1;                     //Kleinster Faktor im Zaehler von
30:     /**/                                        //n!/(k! (n - k)!)
31:     for (i = ergebnis + 1; i <= n; i++){        //Schleife Zaehler
32:       ergebnis *= i;
33:     }
34:     for (i = 2; i <= k; i++){                   //Schleife Zaehler
35:       ergebnis /= i;
36:     }
37:     return ergebnis;
38:   }
39:   public static long binomRekursiv(long n, long k){
40:     if ((n == k) || (k == 0)){                  //Rekursionsverankerung
41:       return 1;
42:     } else {
43:       return binomRekursiv(n - 1, k - 1) +      //Rekursionsschritt
44:         binomRekursiv(n - 1, k);
45:     }
46:   }
47: }
```

14.3 Effektivität

Siehe Lösung von 14.2 Der Vergleich zeigt, daß die rekursive Lösung i. allg. langsamer ist.

15 Arrays

15.1 Eine mathematische Tafel

Listing von **Sources/Array/Arraydef.java**:

```
 1:
 2: public class Arraydef{
 3:   public static void main(String [] args){
 4:     int [] quadrate = new int[10];              //Erzeigen eines leeren Feldes.
 5:     int i;
 6:     for (i = 0; i < 10; i++){                   //Füllen des Arrays
 7:       quadrate[i] = i * i;
 8:     }
 9:     for (i = 0; i < 10; i++){                   //Ausgabe des Arrays
10:       System.out.println(i + ":" + quadrate[i]);
11:     }
12:   }
13: }
```

15.2 Ausgabe in einer Klassenmethode

Siehe 15.3

15.3 Arrays und „Call By Reference“

Listing von **Sources/Array/Arrayref.java**:

```
 1: import corejava.*;
 2:
 3: public class Arrayref{
 4:
 5:   public static void fuelleLinear(int [] arrayRef){
 6:     for (int i = 0; i < arrayRef.length; i++){          //Überschreiben des Arrays
 7:       arrayRef[i] = i;
 8:     }
 9:   }
10:   public static void arrayAusgabe(int [] arrayRef){
11:     for (int i = 0; i < arrayRef.length; i++){
12:       System.out.println(i + ":" + arrayRef[i]);
13:     }
14:   }
15:   public static void main(String [] args){
16:     int [] quadrate = new int[10];                      //Erzeugen eines leeren Feldes.
17:     int i;
18:     for (i = 0; i < 10; i++){                           //Füllen des Arrays
19:       quadrate[i] = i * i;
20:     }
21:     arrayAusgabe(quadrate);
22:     fuelleLinear(quadrate);                             //Übergabe als Referenz.
23:     arrayAusgabe(quadrate);
24:   }
25: }
```

16 Sortieren mittels Quicksort

16.1 Vertauschen zweier Feldelemente eines Arrays

Listing von **Sources/Quicksort/Simplesort.java**:

```
 1: public class Simplesort{
 2:   public static void main(String [] args){
 3:     int [] testArray = new int [8];
 4:     testArray[0] = 23;
 5:     testArray[1] = 3;
 6:     testArray[2] = 2;
 7:     testArray[3] = 11;
 8:     testArray[4] = 3;
 9:     testArray[5] = 6;
10:     testArray[6] = 33;
11:     testArray[7] = 2;
12:     mysort(testArray);
13:     for (int i = 0;
14:          i < testArray.length;
15:          i++){
16:       System.out.print(testArray[i] + " ");
17:     }
18:   }
19:   public static void swap                        //Vertauschen der FeldINHALTE.
20:     (int [] iArray,                              //Das Feld.
21:      int i,                                      //Die Indices der zu
22:      int j){                                     //vertauschhenden Werte.
23:     int help = iArray[i];                        //Zwischenspeichern Position i.
24:     iArray[i] = iArray[j];                       //Position j an Pos. i eintragen.
25:     iArray[j] = help;                            //Alten Wert Pos. i nach Pos. j .
26:   }
27:   public static int getMinIndex                  //Suche des ersten Index eines
28:     (int [] iArray,                              //minimalen Wertes ab Position start
29:      int start){                                 //im Array iArray.
30:     int minIndex = start;                        //Index start als Kandidat.
31:     for (int i = start + 1;                      //Schleifenbeginn nächster Kandidat.
32:          i < iArray.length;                    //Suche bis Arrayende.
33:          i++){
34:       if (iArray[i] < iArray[minIndex]){
35:         minIndex = i;
36:       }
37:     }
38:     return minIndex;
39:   }
40:   public static void mysort(int [] iArray){     //Implementierung eines primitiven
41:     int minIndex;                                //Sortieralgorithmus.
42:     for (int i = 0;
43:          i < iArray.length - 1;
44:          i++){
45:       minIndex = getMinIndex(iArray, i);
46:       swap(iArray, i, minIndex);
47:     }
48:   }
49:
50: }
```

16.2 Implementierung von Quicksort

Listing von **Sources/Quicksort/Quicksort.java**:

```
 1: import corejava.*;
 2:
 3: public class Quicksort{
 4:
 5:   public static void main(String [] args){
 6:     int
 7:       i,
 8:       anzahl = Console.readInt
 9:       ("Wieviele Zahlen wollen Sie eingeben ?");
10:
11:     int [] intArray = new int [anzahl];
```

```
12:
13:     System.out.println("Bitte Zahlen eingeben");
14:     for (i = 0; i < anzahl; i++){
15:       intArray[i] = Console.readInt("Nr. " + (i + 1) + ":");
16:     }
17:     System.out.println("Die Zahlen in aufsteigender Folge lauten:");
18:     intSort(intArray);
19:     for (i = 0; i < anzahl; i++)
20:       System.out.print(intArray[i] + " ");
21:   }
22:   static void cyclicChange                              // Die Funktion cyclicChange
23:     (int [] iArray, int a, int b, int c){               // vertauscht zyklisch die
24:     int temporary = iArray[a];                              // Werte, auf welche die
25:     iArray[a] = iArray[b];                            // Indizes a,b und c zeigen.
26:     iArray[b] = iArray[c];                              // (1,2,3) -> (2,3,1)
27:     iArray[c] = temporary;
28:   }
29:
30:   static void intSort                                   //Hüllfunktion.
31:     (int [] iArray){                                    //Das gesamte Array.
32:     intSortImpl(iArray, 0, iArray.length);              //intSortImpl macht die Arbeit.
33:   }
34:   static void intSortImpl                               //Die "wahre" Sortierfunktion.
35:     (int [] iArray,                                     //Das gesamte Array.
36:      int v,                                             //Startindex des Subarrays.
37:      int laenge){                                       //Länge des Subarrays.
38:     if (1 < laenge){                                    //Abbruchbedingung erreicht?
39:       int teilung = v + laenge / 2;                     //Mittige Teilung.
40:       int centerValue = iArray[teilung];                //Wert in der Mitte.
41:       int p = v;                                        //Untergrenze am Feldanfang.
42:       while (p < teilung){                              //   |p|...|t-1|t|...
43:         if (centerValue < iArray[p]){                   //...|g|...| x |c|...
44:           cyclicChange                                  //...|x|...| c |g|...
45:             (iArray, p, teilung - 1, teilung);              //...|p|....|t|...
46:           teilung--;
47:         }
48:         else
49:           p++;
50:       }
51:       p = v + laenge - 1;                               // Obergrenze Feldende.
52:       while(teilung < p){                               // ...|t|t+1|...|p|...
53:         if (iArray[p] < centerValue){                   // ...|c| x |...|k|...
54:           cyclicChange
55:             (iArray, p, teilung + 1, teilung);              // ...|k| c |...|x|...
56:           teilung++;                                    //   ...| t |...|p|...
57:         }
58:         else
59:           p--;
60:       }
61:       int laengeLinks = teilung - v;                  // Laenge erster Folge.
62:       intSortImpl(iArray, v, laengeLinks);                    // Rekursiver Aufruf
63:       intSortImpl(iArray, teilung + 1,
64:               laenge - laengeLinks - 1);// für beide Teilfolgen.
65:     }
66:   }
67: }
```

17 Eine Klasse zur Repräsentation von Brüchen

17.1 Attribute und Dezimaldarstellung

Listing 17.1 *Eine einfache* `Bruch` *Klasse*
Ouellcode hier verfügbar

```
public class Bruch{
  public double getDezimal(){           //Objektmethode.
    return ((double) zaehler) / nenner; //Cast wegen Integer Divisionsproblematik.
  }
  public int                            //Attribute der Bruchobjekte.
    zaehler,
    nenner;
}
```

Listing 17.2 *Test der* `Bruch` *Klasse*
Ouellcode hier verfügbar

```
import Bruch;                                   //Import der Bruchklasse.

public class Bruchtest{
  public static void main(String [] args){      //Test der Bruchklasse.
    String a = "Hallo";
    Bruch br = new Bruch();                     //Erzeugen eines Bruch Objektes.
    br.zaehler = 2;                             //Initialisieren der Attribute des
    br.nenner = 3;                              //Bruch Objektes.
    System.out.println                          //Methodenaufruf.
      ("Dezimalwert:" + br.getDezimal());
  }
}
```

17.2 Konstruktor und Ausgabe

Listing 17.3 *Erweiterung der* `Bruch` *Klasse*
Ouellcode hier verfügbar

```
public class Bruch{
  public Bruch(){                       //Initialisierung eines Bruchs durch 0/1 .
    zaehler = 0;
    nenner = 1;
  }
  public Bruch(int z){                  //Initialisierung eines Bruchs durch z/1 .
    zaehler = z;
    nenner = 1;
  }
  public Bruch(int z, int n){           //Initialisierung eines Bruchs durch z/n .
    zaehler = z;
    nenner = n;
  }
  public double getDezimal(){
    return ((double) zaehler) / nenner;
  }
  public int getZaehler(){              //get Methode für privates Attribut
    return zaehler;
  }
  public int getNenner(){               //dito.
    return nenner;
  }
  public void ausgabe(){                //Ausgabe des Bruchs auf Bildschirm
    System.out.print                    //als z/n .
      (zaehler + "/" + nenner);
  }
  private int                           //Private (!!) Attribute der Bruchobjekte.
    zaehler,
    nenner;
}
```

Listing 17.4 *Zugriffsschutz privater Attribute*
Quellcode hier verfügbar

```
import corejava.*;
import Bruch;                                   //Import der Bruchklasse.

public class Zugriff{
    public static void main(String [] args){      //Test der Bruchklasse.
    Bruch br = new Bruch();                     //Erzeugen eines Bruch Objektes.
    br.zaehler = 2;                             //Fehler: zaehler ist private.
  }
}
```

Listing 17.5 *Test der Konstruktoren und Methoden*
Quellcode hier verfügbar

```
import corejava.*;
import Bruch;                                   //Import der Bruchklasse.

public class Bruchtest{
  public static void main(String [] args){      //Test der Bruchklasse.
    Bruch
      defaultBr = new Bruch(),                  //Default Konstruktor.
      nurZaehler = new Bruch(4),                //-> 4/1 .
      beides = new Bruch(1,3);                  //-> 1/3 .

    defaultBr.ausgabe();
    System.out.println("");
    nurZaehler.ausgabe();
    System.out.println("");
    beides.ausgabe();
    System.out.println("\nZaehler:" + beides.getZaehler() +
                       " Nenner:" + beides.getNenner());
  }
}
```

18 Brüche und Operatoren

18.1 Addition

18.2 2 Multiplikationen

18.3 Kehrwert

Listing 18.1 *Erweiterung der* `Bruch` *Klasse*
Quellcode hier verfügbar

```
public class Bruch{
  public Bruch(){                      //Initialisierung eines Bruchs durch 0/1 .
    zaehler = 0;
    nenner = 1;
  }
  public Bruch(int z){                 //Initialisierung eines Bruchs durch z/1 .
    zaehler = z;
    nenner = 1;
  }
  public Bruch(int z, int n){          //Initialisierung eines Bruchs durch z/n .
    zaehler = z;
    nenner = n;
    kuerzen();
    if (nenner < 0){
      zaehler *= -1;
      nenner *= -1;
    }
  }
  public Bruch add(Bruch br){
    return new Bruch
      (zaehler * br.nenner + br.zaehler * nenner,
       nenner * br.nenner);
  }
  public Bruch mult(int n){
    Bruch br= new Bruch(n, nenner);
    br.zaehler *= zaehler;
    return br;
  }
  public Bruch mult(Bruch br){
    return new Bruch
      (zaehler * br.zaehler,
       nenner * br.nenner);
  }
  public Bruch kehr(){
    return new Bruch(nenner, zaehler);
  }
  public double getDezimal(){
    return ((double) zaehler) / nenner;
  }
  public int getZaehler(){             //get Methode für privates Attribut
    return zaehler;
  }
  public int getNenner(){              //dito.
    return nenner;
  }
  private void kuerzen(){
    int z = zaehler,
      n = nenner,
      help;
    while (n != 0){                    //Euklidischer Algorithmus zur Bestimmung
      help = z % n;                    //des Größten gemeinsamen Teilers (GGT)
      z = n;                           //von n und z. Nach Verlassen der Schleife
      n = help;                        //enthält z den GGT.
    }
    zaehler /= z;
    nenner /= z;
  }
  public void ausgabe(){               //Ausgabe des Bruchs auf Bildschirm
    System.out.println                 //als z/n .
      (zaehler + "/" + nenner);
  }
  private int                          //Private (!!) Attribute der Bruchobjekte.
```

```
    zaehler,
    nenner;
}
```

Listing 18.2 *Test der* `Bruch` *Klasse*
Ouellcode hier verfügbar

```
import corejava.*;
import Bruch;                                   //Import der Bruchklasse.

public class Bruchtest{
  public static void main(String [] args){      //Test der Bruchklasse.
    Bruch
      zweiDrittel = new Bruch(2, 3),
      dreiSiebtel = new Bruch (3, 7),
      summe = zweiDrittel.add(dreiSiebtel),
      produktZahl = zweiDrittel.mult(7),
      produktBruch = zweiDrittel.mult(dreiSiebtel);
    zweiDrittel.ausgabe();
    dreiSiebtel.ausgabe();
    summe.ausgabe();
    produktZahl.ausgabe();
    produktBruch.ausgabe();
  }
}
```

19 Figuren und Vererbung

19.1 Rechteck und Kreis ohne Vererbung

Listing 19.1 *Die Klasse* `Rechteck`
Quellcode hier verfügbar

```
public class Rechteck
{
  public void figurmittelpunkt                //Erzeugung einer Figur mit
    (double xZent,                            //Mittelpunkt (xZent, yZent).
     double yZent){
    xZentrum = xZent;
    yZentrum = yZent;
  }
  public double getX(){                       //Accessor für privates Attribut
    return xZentrum;                          //xZentrum.
  }
  public double getY(){                       //dito für yZentrum.
    return yZentrum;
  }
  public Rechteck
    (double xZent,                            //Mittelpunktskoordinaten des
     double yZent,                            //Rechteckes.
     double xLen,                             //Länge und Breite des
     double yLen){                            //Rechtecks.
    figurmittelpunkt(xZent, yZent);           //Setzen des Mittelpunkts.
    xLaenge = xLen;                           //Setzen Breite + Länge.
    yLaenge = yLen;
  }
  public String name(){
    return "Rechteck";
  }
  public double getFlaeche(){
    return xLaenge * yLaenge;
  }
  public void skalieren(double faktor){
    xLaenge *= faktor;
    yLaenge *= faktor;
  }
  private double
    xLaenge,
    yLaenge,
    xZentrum,
    yZentrum;
}
```

Listing 19.2 *Die Klasse* `Kreis`
Quellcode hier verfügbar

```
public class Kreis
{
  public void figurmittelpunkt                //Erzeugung einer Figur mit
    (double xZent,                            //Mittelpunkt (xZent, yZent).
     double yZent){
    xZentrum = xZent;
    yZentrum = yZent;
  }
  public double getX(){                       //Accessor für privates Attribut
    return xZentrum;                          //xZentrum.
  }
  public double getY(){                       //dito für yZentrum.
    return yZentrum;
  }
  public Kreis
    (double xZent,                            //Mittelpunktskoordinaten des
     double yZent,                            //Kreises.
     double r){                               //Radius.
    figurmittelpunkt(xZent, yZent);           //Setzen des Mittelpunkts.
    radius = r;                               //Setzen Radius.
  }
```

```
  public String name(){                          //Name der Figur.
    return "Kreis";
  }
  public double getFlaeche(){                    //Fläche der Figur.
    return radius * radius * Math.PI;
  }
  public void skalieren(double faktor){          //Strecken/Stauchen.
    radius *= faktor;
  }
  private double
    radius,
    xZentrum,
    yZentrum;
}
```

Listing 19.3 *Test der Klassen* `Rechteck` *und* `Kreis`
Quellcode hier verfügbar

```
import Rechteck;
import Kreis;

public class Figurtest{
  public static void main(String [] args){
    Rechteck q = new Rechteck(-22, 12, 2, 3);
    System.out.println
      ("Mittelpunkskoordinaten=(" + q.getX() + ","+ q.getY() + ")");
    q.skalieren(3);
    System.out.println("Fläche:" + q.getFlaeche());

    Kreis k = new Kreis(-2, 3, 5);
    System.out.println
      ("Mittelpunkskoordinaten=(" + k.getX() + ","+ k.getY() + ")");
    q.skalieren(3);
    System.out.println("Fläche:" + k.getFlaeche());
  }
}
```

19.2 Rechteck und Kreis mit Vererbung

Listing 19.4 *Die Basisklasse* `Figur`
Quellcode hier verfügbar

```
public abstract class Figur
{
  public Figur                                   //Konstruktor einer Figur mit
    (double xZent,                               //Mittelpunkt (xZent, yZent).
     double yZent){
    xZentrum = xZent;
    yZentrum = yZent;
  }
  public double getX(){                          //Accessor für privates Attribut
    return xZentrum;                             //xZentrum.
  }
  public double getY(){                          //dito für yZentrum.
    return yZentrum;
  }
  public void verschieben                        //Figur um den Vektor (x, y)
    (double x,                                   //verschieben.
     double y){
    xZentrum += x;
    yZentrum += y;
  }
  //Die folgenden, abstrakten Methoden müssen
  //in den konkreten Klassen realisiert
  //werden, siehe z.B. class Quadrat
  public abstract double getFlaeche();           //Fläche der Figur berechnen.
  public abstract void skalieren
    (double faktor);                             //Größe verändern.

  public abstract String name();                 //Kreis,Rechteck,... .
```

```
  private double
    xZentrum,
    yZentrum;
}
```

Listing 19.5 *Die Klasse* `Rechteck`
Ouellcode hier verfügbar

```
import Figur;

public class Rechteck extends Figur
{
  public Rechteck
    (double xZent,
     double yZent,
     double xLen,
     double yLen){
    super(xZent, yZent);                        //Konstruktor Figur(....) Basisklasse.
    xLaenge = xLen;
    yLaenge = yLen;
  }
  public String name(){
    return "Rechteck";
  }
  public double getFlaeche(){
    return xLaenge * yLaenge;
  }
  public void skalieren(double faktor){
    xLaenge *= faktor;
    yLaenge *= faktor;
  }
  private double                                //Zusätzliche Attribute gegenüber der
    xLaenge,                                    //Basisklasse.
    yLaenge;
}
```

Listing 19.6 *Die Klasse* `Kreis`
Ouellcode hier verfügbar

```
import Figur;

public class Kreis extends Figur
{
  public Kreis
    (double xZent,
     double yZent,
     double rad){
    super(xZent, yZent);                        //Konstruktor Figur(....) Basisklasse.
    radius = rad;
  }
  public String name(){
    return "Kreis";
  }
  public double getFlaeche(){
    return radius * radius * Math.PI;
  }
  public void skalieren(double faktor){
    radius *= faktor;
  }
  private double                                //Zusätzliches Attribut gegenüber der
    radius;                                     //Basisklasse.
}
```

Listing 19.7 *Test der Klassen* `Rechteck` *und* `Kreis`
Ouellcode hier verfügbar

```
import Rechteck;
import Kreis;
```

```
public class Figurtest{
  public static void main(String [] args){
    Rechteck r = new Rechteck(10, -3, 3, 5);
    System.out.println("Fläche:" + r.getFlaeche());
    Kreis k = new Kreis(10, -3, 2);
    System.out.println("Fläche:" + k.getFlaeche());
  }
}
```

19.3 Vererbung und Polymorphie

Listing 19.8 *Polymorphe Methodenaufrufe* `getFlaeche()` *und* `name()`
Quellcode hier verfügbar

```
import Rechteck;
import Kreis;

public class Poly{
  public static void main(String [] args){
    Figur f;
    f = new Rechteck(10, -3, 3, 5);
    System.out.println("Fläche:" +              //Polymorpher Aufruf von
                       f.getFlaeche());         //getFlaeche aus class Rechteck.

    System.out.println("Typ:" +                //Polymorpher Aufruf von
                       f.name());               //name() aus class Rechteck.

    f = new Kreis(10, -3, 2);
    System.out.println("Fläche:" +              //Polymorpher Aufruf von
                       f.getFlaeche());         //getFlaeche() aus class Kreis.

    System.out.println("Typ:" +                //Polymorpher Aufruf von
                       f.name());               //name() aus class Kreis.
  }
}
```

20 Bezug auf Basisklassen mittels `super`

20.1 Eigenschaften einer Figur

Listing 20.1 *Die Klasse* `Figur`
Quellcode hier verfügbar

```
public abstract class Figur
{
  public Figur                                  //Konstruktor einer Figur mit
    (double xZent,                              //Mittelpunkt (xZent, yZent).
     double yZent){
    xZentrum = xZent;
    yZentrum = yZent;
  }
  public String eigenschaften(){
    return "Zentrum: (" + xZentrum +
      "," + xZentrum + ")";
  }
  public double getX(){                         //Accessor für privates Attribut
    return xZentrum;                            //xZentrum.
  }
  public double getY(){                         //dito für yZentrum.
    return yZentrum;
  }
  public void verschieben                       //Figur um den Vektor (x, y)
    (double x,                                  //verschieben.
     double y){
    xZentrum += x;
    yZentrum += y;
  }
  public abstract double getFlaeche();          //Fläche der Figur berechnen.
  public abstract void skalieren
    (double faktor);                            //Größe verändern.

  public abstract String name();                //Kreis,Rechteck,... .

  private double
    xZentrum,
    yZentrum;
}
```

Listing 20.2 *Die Klasse* `Rechteck`
Quellcode hier verfügbar

```
import Figur;

public class Rechteck extends Figur
{
  public Rechteck
    (double xZent,
     double yZent,
     double xLen,
     double yLen){
    super(xZent, yZent);                        //Konstruktor Figur(....) Basisklasse.
    xLaenge = xLen;
    yLaenge = yLen;
  }
  public String eigenschaften(){
    return super.eigenschaften() + " Länge:" + xLaenge +
      ", Breite:" + yLaenge;
  }
  public String name(){
    return "Rechteck";
  }
  public double getFlaeche(){
    return xLaenge * yLaenge;
  }
  public void skalieren(double faktor){
    xLaenge *= faktor;
    yLaenge *= faktor;
```

```
  }
  private double                                //Zusätzliche Attribute gegenüber der
    xLaenge,                                    //Basisklasse.
    yLaenge;
}
```

Listing 20.3 *Die Klasse* `Kreis`
Quellcode hier verfügbar

```
import Figur;

public class Kreis extends Figur
{
  public Kreis
    (double xZent,
     double yZent,
     double rad){
    super(xZent, yZent);                        //Konstruktor Figur(....) Basisklasse.
    radius = rad;
  }
  public String eigenschaften(){
    return super.eigenschaften() + " Radius:" + radius;
  }
  public String name(){
    return "Kreis";
  }
  public double getFlaeche(){
    return radius * radius * Math.PI;
  }
  public void skalieren(double faktor){
    radius *= faktor;
  }
  private double                                //Zusätzliches Attribut gegenüber der
    radius;                                     //Basisklasse.
}
```

Listing 20.4 *Polymorpher Test*
Quellcode hier verfügbar

```
import Rechteck;
import Kreis;

public class Poly{
  public static void main(String [] args){
    Figur r = new Rechteck(10, -3, 3, 5);
    System.out.println(r.eigenschaften());
    Figur k = new Kreis(45, 2, 2);
    System.out.println(k.eigenschaften());
  }
}
```

21 Exception Handling

21.1 Der Kehrwert und Ausnahmen

Listing 21.1 *Die Klasse* `Bruch` *mit Ausnahmebehandlung*
Ouellcode hier verfügbar

```
public class Bruch{
  public Bruch(int z, int n) throws ArithmeticException{//Vereinfachte Bruchklasse zur
    if (n == 0){
      throw new ArithmeticException
        ("Bruch(int,int): Nenner ist 0, zaehler =" +
         z);
    }
    zaehler = z;                                        //Demonstration des Exception
    nenner = n;                                         //Handlings.
    if (nenner < 0){
      zaehler *= -1;
      nenner *= -1;
    }
  }
  public double getDezimal(){
    return ((double) zaehler) / nenner;
  }
  public Bruch kehr(){
    return new Bruch(nenner, zaehler);
  }
  private int
    zaehler,
    nenner;
}
```

Listing 21.2 *Test von* `Bruch` *Ausnahmen*
Ouellcode hier verfügbar

```
import corejava.*;
import Bruch;                                  //Import der Bruchklasse.

public class Bruchtest{
  public static void main(String [] args){     //Test der Bruchklasse.
    Bruch
      nullBruch = new Bruch(0, 1);
    try
      {
      Bruch invers = nullBruch.kehr();
      System.out.println("Wert:" + nullBruch.getDezimal());
      }
    catch(ArithmeticException bruchAusnahme)
      {
        System.out.println("Fehler bei Kehrwertbildung:");
        System.out.println(bruchAusnahme);
      }
  }
}
```

21.2 Eine eigene Ausnahmeklasse

Listing 21.3 *Definition der Ausnahmeklasse* `NennerIstNull`
Ouellcode hier verfügbar

```
class NennerIstNull extends ArithmeticException
{
  public NennerIstNull(String msg, int z){
    super(msg);
    zaehler = z;
  }
```

```
  public int getZaehler(){
    return zaehler;
  }
  private int zaehler;
}
```

Listing 21.4 *Die Klasse* `Bruch`
Quellcode hier verfügbar

```
import NennerIstNull;

public class Bruch{
  public Bruch(int z, int n){                          //Vereinfachte Bruchklasse zur
    zaehler = z;                                       //Demonstration des Exception
    nenner = n;                                        //Handlings.
    if (nenner < 0){
      zaehler *= -1;
      nenner *= -1;
    }
  }
  public double getDezimal(){
    return ((double) zaehler) / nenner;
  }
  public Bruch kehr() throws NennerIstNull{
    if (zaehler == 0){
      throw new NennerIstNull("kehr()", nenner);
    }
    return new Bruch(nenner, zaehler);
  }
  private int
    zaehler,
    nenner;
}
```

Listing 21.5 *Test von* `Bruch` *Ausnahmen*
Quellcode hier verfügbar

```
import corejava.*;
import Bruch;                                  //Import der Bruchklasse.

public class Bruchtest{
  public static void main(String [] args){     //Test der Bruchklasse.
    Bruch
      nullBruch = new Bruch(0, 1);
    try
      {
      Bruch invers = nullBruch.kehr();
      System.out.println("Wert:" + nullBruch.getDezimal());
      }
    catch(NennerIstNull br)
      {
        System.out.println("Fehler:" + br);
        System.out.println("Zaehler = " + br.getZaehler());
      }
  }
}
```

22 Hash-Tabellen

22.1 Key und Value vom Typ `String`

Listing 22.1 *Zugriff auf eine Hashtabelle*
Quellcode hier verfügbar

```
import corejava.*;
import java.util.*;

public class Hashtest{

  public static void lookupName(Hashtable h, String s){
    String funktion = (String) h.get(s);
    if (funktion == null){
      System.out.println("Der Name " + s + " ist unbekannt");
    }else{
      System.out.println("Die Funktion von " + s + " ist \"" + funktion + "\"");
    }
  }
  public static void ausgabeHashtabelle(Hashtable h){
    Enumeration e = h.keys();
    String name, funktion;
    while(e.hasMoreElements()){
      name = (String) e.nextElement();
      funktion = (String) h.get(name);
      System.out.println("Mitarbeiter " + name + " hat Funktion \"" + funktion + "\"");
    }
  }
  public static void main(String [] args){
    Hashtable personenFunktion = new Hashtable();
    personenFunktion.put("Schulz", "Versandabteilung");
    personenFunktion.put("Meier", "Poststelle und Mahnwesen");
    personenFunktion.put("Braun", "Auftragserfassung und Erstkontakte");

    lookupName(personenFunktion, "Schulz");
    lookupName(personenFunktion, "Schwarz");

    ausgabeHashtabelle(personenFunktion);

  }
}
```

22.2 Key und Value vom Typ `Integer`

Listing 22.2 *Hashtabelle mit Key und Value aus Integer*
Quellcode hier verfügbar

```
import corejava.*;
import java.util.*;

public class Hashtest{

  public static void lookupInteger(Hashtable h, int key){
    Integer intVal = (Integer) h.get(new Integer(key));
    if (intVal == null){
      System.out.println("Wert zu " + key + " nicht in der Liste");
    } else {
      System.out.println("Der Wert zu " + key + " ist " + intVal);
    }

  }
  public static void ausgabeHashtabelle(Hashtable h){
    Enumeration e = h.keys();
    Integer key, value;
    while(e.hasMoreElements()){
      key = (Integer) e.nextElement();
      value = (Integer) h.get(key);
      System.out.println("Key: " + key + " Wert:" + value);
    }

  }

  public static void main(String [] args)
  {
    Hashtable koordinaten = new Hashtable();
    koordinaten.put(new Integer(-2), new Integer(4));
    koordinaten.put(new Integer(-1), new Integer(1));
    koordinaten.put(new Integer(0), new Integer(0));
    koordinaten.put(new Integer(1), new Integer(1));
    koordinaten.put(new Integer(2), new Integer(4));


    lookupInteger(koordinaten, -2);
    lookupInteger(koordinaten, 10);

    ausgabeHashtabelle(koordinaten);

  }
}
```

23 Ein Server für Primzahlen

23.1 Der Primserver

siehe Lösung zu 23.2

23.2 Der Primserver als Singleton

Implementierung von `Primserver` Listing von **Sources/Primserver/Primserver.java**:

```
 1: import java.util.BitSet;
 2:
 3: public class Primserver{                              //Klasse mit genau einer Instanz
 4:   /**/                                                //gemäß Singleton Design Pattern.
 5:
 6:   public final static Primserver pserv =              //Es wird genau diese einzige In-
 7:     new Primserver(100);                              //stanz erzeugt, siehe Konstr. .
 8:
 9:   public final int getNumPrimes(){                    //Welches ist die Anzahl aller
10:     return primzahlen.length;                         //größten ermittelten Primzahl?
11:   }
12:   public final int getPrimToIndex(int index){        //Zugriff auf Primzahl zu index.
13:     return primzahlen[index];
14:   }
15:   public final int getGroesstePrimzahl(){            //Welches ist die größte bislang
16:     return primzahlen[primzahlen.length - 1];         //ermittelte Primzahl?.
17:   }
18:   public final void constructSiebErathostenes        //Bestimme alle Primzahlen bis
19:     (int obergrenze){                                 //obergrenze.
20:     BitSet isPrim = new BitSet(obergrenze + 1);       //"Das Sieb" .
21:     int i,j;                                          //Allgemeine Laufvariable .
22:     for (i = 2; i <= obergrenze; i++){                //Initialisierung: Annahme, alle
23:       isPrim.set(i);                                  //Zahlen sind Primzahlen(true).
24:     }
25:     int anzahlPrimzahlen = 0;
26:     for (i = 2; i * i <= obergrenze; i++){            //Aussieben.
27:       if (isPrim.get(i)){
28:         anzahlPrimzahlen++;                           //Primzahl gefunden.
29:         for (j = i * i; j <= obergrenze; j += i){     //Vielfache aussortieren.
30:           isPrim.clear(j);                            //Vielfaches keine Primzahl.
31:         }
32:       }
33:     }
34:     do {                                              //Welche Primzahlen fehlen
35:       if (isPrim.get(i)){                             //noch zur Anzahlbestimmung?
36:         anzahlPrimzahlen++;
37:       }
38:     } while (++i <= obergrenze);
39:
40:     primzahlen = new int [anzahlPrimzahlen];          //Array benötigter Größe anlegen.
41:     int primzahlIndex = 0;                            //Start mit Arrayindex 0.
42:     for (i = 2; i <= obergrenze; i++){                //Werte vom Bitset in das
43:       if (isPrim.get(i)){                             //Array der Primzahlen
44:         primzahlen[primzahlIndex] = i;                //übertragen.
45:         primzahlIndex++;                              //aktuellen Arrayindex erhöhen.
46:       }
47:     }
48:   }
49:   private Primserver(int maxPrim)                     //Erzeuge Primz. bis maxPrim.
50:   throws IllegalStateException{                       //Falls noch keine globale In-
51:     if (pserv == null){                               //stanz existiert, erzeuge diese,
52:       constructSiebErathostenes(maxPrim);             //ansonsten Fehlermeldung.
53:     } else {                                          //Es soll keine zweite Instanz
54:       throw new IllegalStateException                 //von Primserver erzeugt werden.
55:         ("Primserver instance already created");
56:     }
57:   }
58:   private int []                                      //Array der ermittelten Prim-
59:     primzahlen;                                       //zahlen, primzahlen[0] == 2,
60: }                                                     //primzahlen[1] == 3, ...   .
```

Implementierung der Testklasse. Listing von **Sources/Primserver/Primservertest.java**:

```
1: public class Primservertest{
2:   public static void main(String argv[]){
3:     for (int i = 0;
4:          i < Primserver.pserv.getNumPrimes();
5:          i++){
6:       System.out.println(Primserver.pserv.getPrimToIndex(i));
7:     }
8:   }
9: }
```

24 Primfaktorzerlegung

24.1 Der Konstruktor

24.2 Weitere Methoden

Listing 24.1 *Implementierung der Primfaktormenge*
Quellcode hier verfügbar

```
import java.util.*;

public class Primfaktormenge{
  public Primfaktormenge(int zahl)
    throws NoPrimeFactors
  {
    if (zahl < 1){
      throw new NoPrimeFactors(zahl);
    } else {
      final int maxPrim =
        Primserver.pserv.getGroesstePrimzahl();
      if (maxPrim * maxPrim < zahl){                    //Falls die größte ermittelte
        Primserver.pserv.constructSiebErathostenes      //Primzahl zu klein ist, muß der
          (1 + (int) Math.sqrt(zahl));                  //Primserver vergrößert werden.
      }
      primfaktorzerlegung(zahl);
    }
  }
  public String toString()
  {                                                     //Methode zur Ausgabe des Ob-
    String result = new String();                       //jektes als Tabelle aus Prim-
    Enumeration allPrimes = primMulti.keys();           //faktoren mit Vielfachheiten.
    while (allPrimes.hasMoreElements()){
      Integer prime = (Integer) allPrimes.nextElement();
      result += "Primfaktor=" + prime +
        ", Vielfachheit=" + (Integer) primMulti.get(prime) + "\n";
    }
    return result;
  }
  public Primfaktormenge ggt                            //Bilde den größten gemeinsamen
    (Primfaktormenge pf){                               //Teiler mit pf.
    Primfaktormenge
      ggtFaktoren = new Primfaktormenge(1);
    Enumeration e = primMulti.keys();
    Integer faktor, thisMulti, pfMulti;
    while(e.hasMoreElements()){
      faktor = (Integer) e.nextElement();
      pfMulti = (Integer) pf.primMulti.get(faktor);
      if (null != pfMulti){
        thisMulti = ((Integer) primMulti.get(faktor));
        int vielfachheit = Math.min
          (pfMulti.intValue(),
           thisMulti.intValue());
        ggtFaktoren.addPrimfaktor
          (faktor.intValue(), vielfachheit);
      }
    }
    return ggtFaktoren;
  }
  public Primfaktormenge kgv                            //Bilde das kleinste gemeinsame
    (Primfaktormenge pf){                               //Vielfache mit pf.
    Primfaktormenge maxteiler = ggt(pf);
    Primfaktormenge thisMinusGgt =
      subtractPrimfaktormenge(maxteiler);
    Primfaktormenge pfMinusGgt =
      pf.subtractPrimfaktormenge(maxteiler);
    return maxteiler.mult(thisMinusGgt).mult(pfMinusGgt);
  }
  public Primfaktormenge mult                           //Bilde das Produkt mit faktor
    (Primfaktormenge faktor){                           //durch Primfaktoraddition.
    Primfaktormenge produkt = new Primfaktormenge(1);
    produkt.addPrimfaktormenge(this);
    produkt.addPrimfaktormenge(faktor);
    return produkt;
  }
```

```
  private Primfaktormenge subtractPrimfaktormenge       //Entferne alle Primfaktoren aus
    (Primfaktormenge subtr){                            //subtr gemäß Vielfachheit.
    Primfaktormenge result = new Primfaktormenge(1);
    Enumeration e = primMulti.keys();
    while(e.hasMoreElements()){
      Integer faktor = (Integer) e.nextElement();
      Integer pMulti = (Integer) subtr.primMulti.get(faktor);
      int vielfachheit =
        ((Integer) primMulti.get(faktor)).intValue();
      if (null == pMulti){
        result.addPrimfaktor
          (faktor.intValue(), vielfachheit);
      } else {
        vielfachheit -= pMulti.intValue();
        if (0 < vielfachheit){
          result.addPrimfaktor(faktor.intValue(), vielfachheit);
        }
      }
    }
    return result;
  }
  private void addPrimfaktormenge                       //Füge Primfaktoren von pf mit
    (Primfaktormenge pf){                               //Vielfachheiten hinzu.
    Integer faktor, thisMult, pfMult;
    Enumeration e = pf.primMulti.keys();
    while(e.hasMoreElements()){
      faktor = (Integer) e.nextElement();
      thisMult = (Integer) primMulti.get(faktor);
      pfMult = (Integer) pf.primMulti.get(faktor);
      if (null == thisMult){
        primMulti.put(faktor, pfMult);
      } else{
        final int vielfachheit = thisMult.intValue() +
          pfMult.intValue();
        primMulti.put(faktor,
                      new Integer(vielfachheit));
      }
    }
  }
  private final void addPrimfaktor                      //Füge den Primfaktor pFaktor
    (int pFaktor,                                       //mit der Vielfachheit multi
     int multi){                                        //zur Zerlegung hinzu.
    Integer faktor = new Integer(pFaktor);
    Integer pMulti =
      (Integer) primMulti.get(faktor);
    if (null == pMulti){
      primMulti.put(faktor, new Integer(multi));
    } else {
      primMulti.put
        (faktor,
         new Integer(pMulti.intValue() + multi));
    }
  }
  private final void primfaktorzerlegung                //Zerlegung von 2 <= zahl in
    (int zahl){                                         //Primfaktoren mit Vielfachheit.
    int
      primZahl,
      index = 0;
    final int anzPrim = Primserver.pserv.getNumPrimes();//Anzahl d. Primz. des Servers.
    do {                                                //Schleife über Primzahlen so-
      primZahl = Primserver.pserv.getPrimToIndex(index);//lange deren Quadrat kleiner
      index++;                                          //zahl ist.
      if (zahl % primZahl == 0){                        //Falls Rest == 0, dann wurde
        int multi = 0;                                  //ein Primfaktor gefunden.
        do{
          zahl /= primZahl;
          multi++;
        } while  (zahl % primZahl == 0);
        addPrimfaktor(primZahl, multi);
      }
    } while (primZahl * primZahl < zahl &&              //Abbruchbedingung: zahl unzer-
             index < anzPrim);                          //legt oder letzte Primzahl.
    if (1 < zahl){                                      //Ist zahl selbst noch eine
      addPrimfaktor(zahl, 1);                           //Primzahl?
    }
  }
  private Hashtable primMulti = new Hashtable();
}
```