Algorithmen und Datenstrukturen

Stichworte und allgemeine Informationen zur Lehrveranstaltung (LV)
Algorithmen und Datenstrukturen (B-MI2-ALG) für den **Zug 1**
im Studiengang **Bachelor Medieninformatik** (B-MI)
im **Wintersemester 2017/18** bei Ulrich Grude.

Anmerkung: Die parallele Veranstaltung für den **Zug 2** wird von **Prof. Böhler** betreut.

In der vorliegenden Datei finden Sie u.a. die Termine der einzelnen seminaristischen Unterrichte (SUs) und Übungen (Üs), die Regeln, nach denen man für diese LV Noten bekommt und weitere Informationen. Im Laufe des Semesters können Sie in dieser Datei nach jedem SU ein paar **Stichworte zum behandelten Stoff** (und manchmal auch Korrekturen und Ergänzungen etc.) finden.

Einleitung

1. Diese LV stellt ziemlich hohe Ansprüche an ihre TeilnehmerInnen. Es geht darin unter anderem um *rekursive Programmierung* (deren Einfachheit und Eleganz vielen Studierenden erst nach intensivem Üben aufgeht), um die *Komplexität von Algorithmen* (z.B. um den Unterschied zwischen $O(n^2)$ und $O(2^n)$), um größere Zahlen (Wie viele Nullen hat 1 Billion? How many zeroes has 1 billion?) und um eines der wichtigsten und schwierigsten Grundlagenprobleme der Informatik (Ist P gleich NP oder nicht?).

2. Diese LV kann interessant und nützlich sein und erfolgreich abgeschlossen werden, wenn folgende Voraussetzungen erfüllt sind:

- Sie haben sich fest vorgenommen, an allen SUs und Üs teilzunehmen
  (und wenn Sie dann ausnahmsweise doch mal etwas verpassen holen Sie es sofort gründlich nach).
- Sie können regelmäßig donnerstags um 10 Uhr an einem kleinen Test teilnehmen (auch bei Glatteis!).
- Sie können und wollen in kleinen Gruppen (zu zweit, höchstens zu dritt) zusammenarbeiten.

Wenn diese Voraussetzungen nicht alle erfüllt sind, sollten Sie diese LV wahrscheinlich nicht belegen (sonst machen Sie sich, anderen Studierenden und dem Betreuer der LV möglicherweise nutzlos Arbeit und Ärger).

3. Auf den folgenden Seiten werden
- die Organisation dieser LV und
- die Regeln, nach denen man Noten bekommt

beschrieben. Lesen Sie die Regeln (insbesondere S. 3 und 4) genau durch, am besten bevor Sie die LV belegen.

Einige Unterlagen zu dieser LV finden Sie auf meiner Netzseite (ziemlich weit oben)

http://public.beuth-hochschule.de/~grude/

Außerdem brauchen TeilnehmerInnen an dieser LV Zugriff auf das Buch **"Java ist eine Sprache"** von Ulrich Grude. Aus dem Netz der Beuth Hochschule können Sie kostenlos auf die Online-Version https://link.springer.com/book/10.1007/978-3-322-80263-7 zugreifen. Ein Papier-Exemplar können Sie (ebenfalls kostenlos) in der Bibliothek der Beuth Hochschule ausleihen.

Termine dieser LV

Erster Termin: Donnerstag **12.10.2017, 08.00 Uhr, Raum D E17** (SWE-Labor, Eingang durch Raum **D E16b**, Verlosung der Übungsplätze, danach Übung im Raum **D 113**, LIS-Labor)

Termine der seminaristischen Unterrichte (SUs) und Übungen (Üs):

KW	SW	Donnerstag	Block 1: Ü1a	Block 2: SU	Block 3: Ü1b	Block 4: Ü1c	Vorführung von Projekten
41	01	12.10.2017	Ü1a-01	SU-01	Ü1b-01	Ü1c-01	
42	02	19.10.2017	Ü1a-02	T01, SU-02	Ü1b-02	Ü1c-02	
43	03	26.10.2017	Ü1a-03	T02, SU-03	Ü1b-03	Ü1c-03	Projekt 1
44	04	02.11.2017	Ü1a-04	T03, SU-04	Ü1b-04	Ü1c-04	
45	05	09.11.2017	Ü1a-05	T04, SU-05	Ü1b-05	Ü1c-05	Projekt 2
46	06	16.11.2017	Ü1a-06	T05, SU-06	Ü1b-06	Ü1c-06	
47	07	23.11.2017	Ü1a-07	T06, SU-07	Ü1b-07	Ü1c-07	Projekt 3
48	08	30.11.2017	Ü1a-08	T07, SU-08	Ü1b-08	Ü1c-08	
49	09	07.12.2017	Ü1a-09	T08, SU-09	Ü1b-09	Ü1c-09	Projekt 4
50	10	14.12.2017	Ü1a-10	T09, SU-10	Ü1b-10	Ü1c-10	
51	11	21.12.2017	Ü1a-11	T10, SU-11	Ü1b-11	Ü1c-11	Projekt 5
52 01		23.12.17 - 03.01.18	Ferien				
01	12	04.01.2018	Ü1a-12	T11, SU-12	Ü1b-12	Ü1c-12	
02	13	11.01.2018	Ü1a-13	T12, SU-13	Ü1b-13	Ü1c-13	Projekt 6
03	14	18.01.2018	Ü1a-14	T13, SU-14	Ü1b-14	Ü1c-14	
04	15	25.01.2018	Ü1a-15	SU-15	Ü1b-15	Ü1c-15	
05	16	01.02.2018		Klausur?			
06	17	08.02.2018		Rückgabe?			

Aküs

Aküs: Abkürzungen
KW: Kalenderwoche
SW: Semesterwoche

SU: seminaristischer Unterricht
Ü1a: Übungsgruppe 1a
Ü1b: Übungsgruppe 1b
Ü1c: Übungsgruppe 1c

T03: Test Nr. 3
SU-03: SU, 3. Termin
Ü1a-03: Ü1a, 3. Termin
Ü1b-03: Ü1b, 3. Termin
Ü1c-03: Ü1c, 3. Termin

Belegungszeitraum WS17/18: 15.09.2017 - 18.10.2017

Organisation dieser Lehrveranstaltung (LV)

1. **Übungsgruppen und Arbeitsgruppen**: Eine *Übungsgruppe* besteht aus etwa 20 Personen, die zu den gleichen Zeiten Übungen haben. Für den Zug 1 sind drei Übungsgruppen geplant (1a,1b und 1c).

Eine *Arbeitsgruppe* besteht aus 2 Personen, die
- zur gleichen Übungsgruppe gehören und
- einen Gruppen-Namen gewählt haben (z.B. *Los Amigos* oder *Die Gummiadler* oder *ABC* ... etc.) und
- sich per Email bei mir als Arbeitsgruppe angemeldet haben.

In Ausnahmefällen kann eine Arbeitsgruppe auch aus 3 Personen bestehen (sprechen Sie mit mir).

Nur Arbeitsgruppen können *Projekte vorführen* und per Email an mich *Fragen stellen* (Einzelpersonen dürfen in den SUs und Üs Fragen stellen, aber nicht per Email). Mit dieser Regelung wird folgendes angestrebt:
- Sie sollen üben, in Gruppen zu arbeiten (eine wichtige Fähigkeit für alle InformatikerInnen).
- Meine Arbeit soll effektiver gemacht werden (statt 2 Emails möchte ich nur 1 Email beantworten).

2. **Übungen**: Jede Übungsgruppe hat pro Woche einen Block Übung. Am Anfang einer Übung wird jeweils "etwas Wissenswertes" behandelt. Fragen dazu können im nächsten Test drankommen. In der restlichen Übungszeit können Sie Fragen zu den 6 Projekten (P01 bis P06) stellen und Projekte bearbeiten und vorführen. Bei der Vorführung eines Projekts müssen alle Mitglieder der Arbeitsgruppe anwesend und bereit sein, Fragen zu dem Projekt zu beantworten. Sie sollten
- sich auf die **Übungen vorbereiten**,
- den Text zum aktuellen Projekt (in der Datei **Projekte.odt**) *vorher* durcharbeiten (und nicht erst während der Übung zum ersten Mal ansehen),
- schon möglichst viel selbständig (in Ihrer Arbeitsgruppe) programmieren und
- Ihre Fragen zum aktuellen Projekt in schriftlicher Form in die Übungen mitbringen.

3. **Tests**: Am Anfang der meisten SUs (d.h. am Anfang des 2. Blocks) wird ein kleiner Test geschrieben, insgesamt 13 Stück (T01 bis T13, siehe oben S. 2), Bearbeitungszeit jeweils etwa 10 bis 15 Minuten. Die Aufgaben und Fragen in diesen Tests werden sich auf den Stoff beziehen, der (in den SUs und in den Üs) vorher behandelt wurde. In jedem Test kann man maximal 10 Punkte erreichen. Mit 5 oder mehr Punkten hat man den Test *bestanden*, mit weniger Punkten hat man den Test *nicht bestanden*. Zu jedem Test dürfen Sie als Unterlage 1 Blatt (max. DIN A 4, beliebig beschriftet) mitbringen. Ein "Nachschreiben" von verpassten Tests ist ausgeschlossen. Wenn Sie weniger als 10 der Tests bestehen, sind Sie durchgefallen (d.h. Sie bekommen die Modul-Note 5,0. Siehe dazu auch den nächsten Abschnitt auf S. 4).

Achtung: Wenn Sie an einem Test nicht teilgenommen haben, dann macht es **keinen Unterschied**, *warum* Sie nicht nicht teilgenommen haben (weil Sie krank waren, weil Sie keine Lust hatten, weil die S-Bahn nicht pünktlich fuhr etc.). Ein ärztliches Attest oder eine andere Bescheinigung kann Ihnen also nicht dabei helfen, diese LV zu bestehen. Andererseits müssen Sie nicht alle Tests bestehen, sondern nur 10 von 13 (Sie können also z.B. einen Test wegen Krankheit und einen wegen einer eingefrorenen S-Bahn verpassen und trotzdem noch eine gute oder sogar sehr gute Note erreichen).

Dringende Empfehlung: Nehmen Sie an allen Tests teil, bei denen Ihnen das irgendwie möglich ist. Sparen Sie die 3 Tests, an denen Sie *nicht* unbedingt teilnehmen müssen, für Unvorhergesehenes auf (für Krankheit, eine unpünktliche S-Bahn etc.).

4. **Software**: Die Projekte sollten normalerweise mit **Eclipse** bearbeitet werden. Um einige der Unterlagen (z.B. die Datei **Projekte.odt**) lesen zu können, braucht man **OpenOffice** oder **LibreOffice** (siehe https://www.openoffice.org/de/ bzw. https://de.libreoffice.org/).

Tipp: Eclipse bearbeitet jeweils einen *Workspace*. Diesen Workspace kann man (nicht nur auf einer Festplatte sondern auch) auf einem **USB-Stick** einrichten. Die Programme im Workspace auf dem Stick kann man dann mit verschiedenen Rechnern (auf denen Eclipse installiert ist) bearbeiten.

Wie bekommt man Noten für diese LV?

Für diese LV kann man 3 Noten bekommen: Eine **Übungs-Note**, eine **Klausur-Note** und eine **Modul-Note**. Nur die Modul-Note wird später auf Ihren Studienbescheinigungen erscheinen, die Übungs-Note und die Klausur-Note dienen nur dazu, die Modul-Note zu berechnen.

Ihre Übungs-Note wird aus Ihren Übungs-Punkten berechnet. Deren Anzahl können Sie durch die Teilnahme an den Tests vergrößern und durch zu-spätes Vorführen von Projekten verkleinern. Es folgen hier die genauen Regeln, nach denen Sie Übungs-Punkte, eine Übungs-Note und eine Modul-Note bekommen:

1. Wenn Sie weniger als 10 Tests bestehen sind Sie durchgefallen (Modul-Note 5,0).
2. Sonst zählen ihre 10 besten Testergebnisse als Übungs-Punkte
(das sind mindestens 50 und höchstens 100 Punkte).
3. Wenn Ihre Arbeitsgruppe bis zum **Do 18.01.2018** nicht für jedes der 6 Projekte eine akzeptable Lösung vorgeführt hat, sind alle Mitglieder der Gruppe durchgefallen (Modul-Note 5,0).
4. Wenn Ihre Arbeitsgruppe ein Projekt n Wochen verspätet vorführt, werden jedem Gruppenmitglied dafür 5*n Übungs-Punkte abgezogen (z.B. werden für 2 Wochen Verspätung jedem Gruppenmitglied 10 Übungs-Punkte abgezogen).

Aus der Anzahl Ihrer Übungs-Punkte wird nach folgender Tabelle ihre Übungs-Note berechnet:

Punkte (ab):	95	90	85	80	75	70	65	60	55	50	0-49
Note:	1,0	1,3	1,7	2,0	2,3	2,7	3,0	3,3	3,7	4,0	5,0

5. In der Klausur am Ende es Semesters sind 100 Punkte erreichbar. Die Klausur-Note wird auch nach obiger Tabelle berechnet. Das Gleiche gilt für die Nachklausur (kurz vor Beginn des SS18).

6. In die Modul-Note geht die Klausur-Note mit 75 % und die Übungs-Note mit 25 % ein. Bei der Berechnung der Modul-Note wird in Zweifelsfällen zu Ihren Gunsten gerundet, z.B. so:

Übungs-Note	**Klausur-Note**	**Rohergebnis**	**Modul-Note**
2,3	2,7	(2,3 + 3 * 2,7) / 4 = 2,60	2,7
3,7	2	(3,7 + 3 * 2,0) / 4 = 2,425	2,3
2,0	4,0	(2,0 + 3 * 4,0) / 4 = 3,50	3,3

Ü-01, Do 12.10.2017, Block 1, 3, 4

1. Kurze Begrüßung

2. Jede Teilnehmerin sollte versuchen, sich auf einem Labor-Rechner einzuloggen (für den Fall, dass Sie etwas ausdrucken wollen, oder dass Ihr Laptop mal kaputt ist, oder ...)

3. Am Anfang jeder Übung werden wir kurz (ca. 10 min) etwas "Wissenswertes" besprechen. Der besprochene Stoff kann im nächsten Test abgefragt werden.

4. Mit der Bezeichnung **"das Buch"** ist in dieser Datei immer das Buch "Java ist eine Sprache" von Ulrich Grude gemeint.

Heute geht es um folgende Fragen:

5.1. Ungefähr wie viele Werte gehören (in der Programmiersprache Java) zum Typ `int`?

Ungefähr 4.3 Milliarden.
Englisch: About 4.3 billion

Achtung: Bruchzahlen werden hier (wie in England, den USA und in Java-Programmen üblich) mit einem Dezimal**punkt** notiert, nicht mit einem Dezimal**komma** (wie in Deutschland üblich).

5.2. Ungefähr in welchem Zahlenbereich (von? bis?) liegen diese `int`-Werte?

Ungefähr im Bereich von -2.15 Milliarden bis +2.15 Milliarden.

5.3. Ungefähr wie viele Komponenten kann eine Reihung (an array) in Java höchstens haben?

Ungefähr 2.15 Milliarden

5.4. Ungefähr wie viele Werte gehören (in Java) zum Typ `long`?

Ungefähr 18.4 Trillionen
Englisch: About 18.4 quintillion

5.5. Ungefähr in welchem Zahlenbereich (von? bis?) liegen diese `long`-Werte?

Ungefähr im Bereich von -9.2 Trillionen bis +9.2 Trillionen

5.6. Ungefähr wie viele `char`-Komponenten kann ein `String`-Objekt enthalten?

Ungefähr 2.15 Milliarden `char`-Komponenten.

5.7. Wie viele Zeichen enthält ein String der Länge 10?

Mindestens 5 und höchstens 10 (weil häufig gebrauchte Zeichen durch je *einen* `char`-Wert dargestellt werden, aber selten gebrauchte Zeichen durch je *zwei* `char`-Werte). Siehe dazu auch im Buch das Kapitel **23 Java und der Unicode** ab S. 558.

Heute (in der 1. Übung des Semesters) besprechen wir noch mehr Stoff (statt "zu üben"):

Notation für Bereiche von Ganzzahlen

Zum abgeschlossenen	Bereich [3, 7] gehören die Zahlen	3, 4, 5, 6, 7
Zum offenen	Bereich (3, 7) gehören die Zahlen	4, 5, 6
Zum halboffenen	Bereich [3, 7) gehören die Zahlen	3, 4, 5, 6
Zum halboffenen	Bereich (3, 7] gehören die Zahlen	4, 5, 6, 7

Reihungen (engl. arrays) und Sammlungen

Beispiele:

```
1 int[]             revi = new int[100];             // eine Reihunge
2 ArrayList<String> alof = new ArrayList<String>();  // eine Sammlung
3 ArrayList<String> alof = new ArrayList<>();        // seit Java 8
```

Positive Eigenschaften von Reihungen im Vergleich zu Sammlungen?
- die Komponenten können zu einem Referenztyp oder zu einen *primitiven Typ* gehören
- der Zugriff auf eine Komponente geht sehr schnell

Eine negative Eigenschaften von Reihungen:
- Die Länge einer Reihung kann nicht verändert werden ("Reihungen sind aus Beton").

Angenommen, wir wollen in einem Java-Programm ab und zu einen `int`-Wert in die Reihung `revi` einfügen. Wie können wir verhindern, dass wir beim Einfügen eines weiteren Wertes schon eingefügte Werte überschreiben?

Lösung: Wir vereinbaren eine `int`-Variable namens `lbi` und speichern darin immer den **letzten belegten Index**. Mit welchem Wert sollten wir `lbi` initialisieren?

```
4 int lbi = -1; // -1 ist gar kein Index, aber anfangs "der zuletzt belegte Index"
```

Mit welchem Befehl können wir jetzt den Wert einer Variablen `n` in die Reihung `revi` einfügen?

```
5 revi[++lbi] = n; // lbi wird zuerst erhöht, dann wird n eingefügt
```

Anmerkung: Der Befehl `++` wurde extra dafür erfunden, damit man das Einfügen eines Wertes in eine Reihung mit einem einzigen Befehl erledigen kann, und nicht 2 Befehle dafür braucht.

Welche Komponente der Reihung kann man besonders leicht löschen, und mit welchem Befehl?

Die Komponente mit dem Index `lbi` kann man mit folgendem Befehl löschen:

```
6 lbi--; // oder genauso gut: --lbi;
```

Die Reihung `revi` brauchen wir *nicht zu verändern*, denn "wir nehmen ja nur die Komponenten ernst, die Indizes zwischen 0 und `lbi` haben". Komponenten, die nicht belegt sind (d.h. Komponenten mit Indizes größer als `lbi`) "existieren logisch gar nicht" (sie existieren nur physikalisch).

Angenommen, `i` ist ein Index aus dem Bereich `[0, lbi]`.
Mit welchem Befehl können wir die Komponente mit dem Index `i` (d.h. `revi[i]`) löschen?

```
7 revi[i] = revi[lbi--];
```

Diese Zuweisung hat zwei Wirkungen:

1. Sie kopiert den Wert `revi[lbi]` an die Stelle `revi[i]`.
2. Danach löscht sie die Komponente mit dem Index `lbi` (die ja besonders leicht zu löschen ist).

SU-01, Do 12.10.2017, Block 2

1. Begrüßung

2. Die Datei Stichworte.pdf

Wer war schon mal auf meiner Netzseite und hat sich diese Datei angesehen?
Wer noch nicht hat: Möglichst bald nachholen.

Hier eine Kurzversion der wichtigsten Regeln (Genaueres steht in der Datei):

1. Am Anfang des SU (also donnerstags um 10 Uhr) werden wir meistens (13 Mal) einen kleinen Test schreiben. Davon müssen Sie mindestens 10 bestehen, sonst sind Sie *durchgefallen* (Modulnote 5,0). Für bestandene Tests bekommen Sie **Pluspunkte**.

2. Zu jedem Test dürfen Sie als Unterlage 1 Blatt, maximal DIN A 4, *beliebig beschriftet*, mitbringen.

Tipp: Kommen Sie immer 10 oder 15 Minuten vor dem Beginn eines Tests (nicht 5 Minuten danach).

3. Wenn Sie an einem Test nicht teilnehmen, haben Sie ihn nicht bestanden. Warum Sie nicht teilgenommen haben (Krankheit, S-Bahn fuhr nicht, hatte keine Lust etc.) spielt keine Rolle.

4. Für die Übungen sollten Sie zu zweit (höchstens zu dritt) eine **Arbeitsgruppe** bilden. Diese Arbeitsgruppe sollte (im Laufe des Semesters) 6 Projekte bearbeiten und das Ergebnis vorführen. Wenn eine Gruppe nicht für jedes Projekt eine akzeptable Lösung vorführt, sind alle Gruppen-Mitglieder *durchgefallen* (Modulnote 5,0). Wenn eine Gruppe ein Projekt zu spät vorführt, bekommen die Mitglieder dafür **Minuspunkte** (5 pro Woche Verspätung).

5. Aus den Pluspunkten (für die Tests) und den Minuspunkten (für verspätet vorgeführte Projekt-Lösungen) wird eine **Übungsnote** berechnet.

6. Am Ende des Semesters wird eine Klausur geschrieben. Dafür bekommen Sie eine **Klausurnote**.

7. In die Berechnung Ihrer **Modulnote** geht die Übungsnote mit 25 % und die Klausurnote mit 75 % ein.

8. Am Anfang jeder Übung wird kurz etwas **Wissenswertes** behandelt. Dieser Stoff kann dann im nächsten Test abgefragt werden. Kommen Sie also auch zu den Übungen möglichst immer pünktlich.

Empfehlung: Gehen Sie zuerst einmal davon aus, dass diese LV "Ihren vollen Einsatz" erfordert.

3. Namen und Fotos

Ich würde gern Ihre Namen lernen, um Sie damit anreden zu können (z.B. Frau Herter oder Herr Gart, nicht: Sie da drüben, nein, daneben, ...). Bitte unterstützen Sie mich dabei, indem Sie mir noch **in dieser Woche** per Email ein Foto von sich schicken, auf dem Ihr Gesicht gut erkennbar ist (also möglichst ohne große Sonnenbrille). Nennen Sie die Bilddatei bitte entsprechend den folgenden Beispielen:

Herter-Kristina.jpg oder **Gart-Igor.png** (also **Nachname-Vorname.jpg** oder **.png**).

Von folgenden Personen habe ich bereits ein Foto:

Aktas, Bieck, Brett, Cavusoglu, Dunker, Herrmann, Karabulut, Meier.

4. Bis nächste Woche

Bitte sehen Sie sich die Regeln in der Datei **Stichworte.pdf** genau an.
Wenn Sie dann noch Fragen dazu haben, können wir die nächste Woche besprechen.
Den 1. Test schreiben wir nächsten Donnerstag um 10 Uhr.

Hat jemand noch eine Frage zu dieser LV die unbedingt noch heute besprochen werden sollte?

Dann geht es jetzt los mit dem Stoff dieser LV.

Rekursion

Aufgabe: Schreiben Sie eine (Java-) Methode entsprechend der folgenden Spezifikation:

```
 1    static void mpi(String s, int anz) { // mehrmals print, iterativ
 2       // Macht nichts, wenn anz kleiner als 1 ist.
 3       // Gibt sonst s anz-mal aus.
 4
 5       for (int i=1; i<=anz; i++) {
 6          printf("%s", s);
 7       }
 8    }
```

Versuchen Sie, die gleiche Aufgabe *ohne eine Schleife* zu lösen.

Lösung ohne Schleife:

```
 9    static void mpr(String s, int anz) { // mehrmals print, rekursiv
10       // Macht nichts, wenn anz kleiner als 1 ist.
11       // Gibt sonst s anz-mal aus.
12
13       if (anz<1) return;
14       printf("%s", s);
15       mpr(s, anz-1);
16    }
```

Aufgabe: Schreiben Sie eine Methode entsprechend der folgenden Spezifikation:

```
17    static int faku(int n) {
18       // Liefert 1, wenn n kleiner/gleich 1 ist, und sonst
19       // das Produkt aller Ganzzahlen im Bereich [1, n].
20
21       if (n<=1) return 1;
22       return n * faku(n-1);
23    }
```

Grundlagen zum Thema Rekursive Unterprogramme

Dieser Abschnitt gilt für *alle Programmiersprachen* (nicht nur für Java). Deshalb verwenden wir hier den allgemeinen Begriff *Unterprogramm* (und nicht den spezielleren Begriff *Methode*, der nur im Zusammenhang mit objektorientierten Sprachen üblich ist).

Def.: Ein Unterprogramm ist *rekursiv*, wenn es sich (direkt oder indirekt) selbst aufruft.

Direkt rekursiv: Ein Unterprogramm `u` ist direkt rekursiv, wenn sein Rumpf (mindestens) einen Aufruf von `u` enthält.

Beispiele: Die Funktion `faku` und die Prozedur `mpr` (sieh oben) sind *direkt rekursiv.*

Indirekt rekursiv: Wenn ein Unterprogramm
`u1` ein Unterprogramm `u2` aufruft, und
`u2` ein Unterprogramm `u3` aufruft, und
...
`u16` ein Unterprogramm `u17` aufruft, und
`u17` wieder das Unterprogramm `u1` aufruft,
dann ist `u1` *indirekt rekursiv* (und das Gleiche gilt für `u2` bis `u17`).

Allgemeine Form aller rekursiven Unterprogramme

1. Der Rumpf eines rekursiven Unterprogramms muss aus einer *Fallunterscheidung* bestehen.
Häufig ist das eine `if`-Anweisung, es kann aber auch eine `switch`-Anweisung oder ein ähnliches Konstrukt sein (je nach Programmiersprache).

2. Die *Fallunterscheidung* muss unterscheiden zwischen
einfachen Fällen (in denen sich das Unterprogramm *nicht* wieder aufruft) und
rekursiven Fällen (in denen sich das Unterprogramm *wieder aufruft*).

3. Jedes rekursive Unterprogramm muss
mindestens *einen* einfachen Fall und
mindestens *einen* rekursiven Fall enthalten.

Frage-01: Was wäre mit einem Unterprogramm, welches *keinen einfachen Fall* enthält?
(Es würde zu einer *Endlosrekursion* führen).

Frage-02: Was wäre mit einem Unterprogramm, welches *keinen rekursiven Fall* enthält?
(Es wäre nicht rekursiv).

Grundregeln zum Schreiben einer rekursiven Methode:

Angenommen, Sie wollen eine rekursive Funktion wie
`static int rm (int n ) { ... }`
schreiben.

1. Machen Sie sich vertraut damit, was die Methode `rm` machen soll, und üben Sie,
das präzise zu beschreiben (indem Sie es z.B. einer KollegIn erklären).

2. Erinnern Sie sich daran, dass der Rumpf der Methode eine *Fallunterscheidung* sein muss
(Sie können also meistens schon mal "`if` " hinschreiben :-)).

3. Empfehlung: Behandeln Sie im Rumpf von `rm` zuerst die *einfachen Fälle*.

4. Wenn Sie danach die *rekursiven Fälle* behandeln, gilt: n ist kein einfacher Fall.
Beispiel: Für die Funktion `faku` bedeutet das konkret: `n` ist größer als `1`.

5. Sie dürfen (im Rumpf von `rm`) die Methode `rm` beliebig oft (rekursiv) aufrufen,
aber *nie* mit dem Parameter `n` als Argument:

```
40  static int rm(int n) { // Parameter n
41     // Zuerst werden die einfachen Faelle behandelt:
42     ...
43     // Dann werden die rekursiven Faelle behandelt:
44     ...
45     ... rm(n) ... // VERBOTEN: Rekursiver Aufruf mit n als Argument.
46     ...
```

**6. Als Argument (in einem rekursiven Aufruf von rm) ist nur ein Ausdruck erlaubt,
dessen Wert *näher bei einem einfachen Fall liegt* als der Wert von n**
(und das gilt offenbar *nicht* für n selbst).

Wenn z.B. alle Argumente, die kleiner/gleich `1` sind als einfache Fälle behandelt wurden (wie oben in der rekursiven Funktion `faku`), dann sind unter anderem folgende Argumente in rekursiven Aufrufen erlaubt:
`n-1` (weil `n-1` näher bei `1` liegt als `n`)
`n-5` (weil `n-5` selbst ein einfacher Fall ist oder näher bei einem solchen liegt als `n`)
`n/2` (weil `n/2` näher bei `1` liegt als `n`)
`n/5` (weil `n/5` näher bei `1` liegt als `n`)

Nicht erlaubt wären in einem rekursiven Aufruf z.B. die Argumente `n+1`, `n+5`, `n*2`, `n*5` etc., weil sie weiter weg von den einfachen Fällen liegen als `n`.

Ü-02, Do 19.10.2017, Block 1, 3, 4

```
 1 // Datei Wiwe01_printfGanz.java
 2 /* ------------------------------------------------------------------------
 3 In diesem Programm werden Ganzzahlen verschiedener Typen mit dem printf-
 4 Befehl auf unterschiedliche Weisen formatiert und zum Bildschirm ausgegeben.
 5 ------------------------------------------------------------------------ */
 6 public class Wiwe01_printfGanz {
 7    // ------------------------------------------------------------------
 8    // Wir vereinbaren den (relativ kurzen) Namen printf als Abkuerzung fuer
 9    // den (unpraktisch langen) Namen System.out.printf:
10    static void printf(String f, Object... v) {System.out.printf(f, v);}
11    // Die Methode printf hat 1 Parameter vom Typ String und 0 oder mehr
12    // ("beliebig viele") Parameter vom Typ Object. v ist vom Typ Object[].
13    // ------------------------------------------------------------------
14    // Der 1. Parameter heisst FORMAT-STRING. Er kann Umwandlungsbefehle
15    // enthalten wie z.B. %d oder %5d oder %+,d oder %s oder %10s etc.
16    // Fuer jeden Umwandlungsbefehl im Format-String muss (nach dem
17    // Format-String) ein weiterer Parameter p angegeben werden.
18    // 2 Ausnahmen (OHNE weiteren Param.): %n bezeichnet einen Zeilenwechsel,
19    // %% bezeichnet ein (nicht zwei!) Prozentzeichen.
20    // ------------------------------------------------------------------
21    static public void main(String[] sonja) {
22       printf("Wiwe01_printfGanz: Jetzt geht es los!%n");
23       printf("-----------------------------------%n");
24       // Ganzzahl-Variablen verschiedener Typen:
25       byte       v01 =    Byte.MAX_VALUE;
26       short      v02 =   Short.MAX_VALUE;
27       int        v03 = Integer.MAX_VALUE;
28       long       v04 =    Long.MAX_VALUE;
29
30       // In einer Mindestbreite von 19 Zeichen ausgeben (mit ' ' polstern):
31       printf("A v01: %19d%n", v01);
32       printf("B v02: %19d%n", v02);
33       printf("C v03: %19d%n", v03);
34       printf("D v04: %19d%n", v04);
35       printf("-----------------------------------%n");
36       // In einer Mindestbreite von 19 Zeichen ausgeben (mit '0' polstern):
37       printf("E v01: %019d%n", v01);
38       printf("F v02: %019d%n", v02);
39       printf("G v03: %019d%n", v03);
40       printf("H v04: %019d%n", v04);
41       printf("-----------------------------------%n");
42       // Mindestbreite 25, Ziffern ("landesueblich") gruppiert:
43       printf("I v01: %,25d%n", v01);
44       printf("J v02: %,25d%n", v02);
45       printf("K v03: %,25d%n", v03);
46       printf("L v04: %,25d%n", v04);
47       printf("-----------------------------------%n");
48       // Negative Zahlen beginnen mit '-', positive haben kein Vorzeichen:
49       printf("M +100: %d%n", +100);
50       printf("N -200: %d%n", -200);
51       printf("-----------------------------------%n");
52       // Negative Zahlen beginnen mit '-', positive mit '+':
53       printf("O +100: %+d%n", +100);
54       printf("P -200: %+d%n", -200);
55       printf("-----------------------------------%n");
56       int z1=12, z2=520, z3=50, z4=77;               // Ein paar Zahlen
57       printf("Q %5d Euro und %2d Cent%n", z1, z3);
58       printf("R %5d Euro und %2d Cent%n", z2, z4);
59       printf("-----------------------------------%n");
60       String t1 = "Franken", t2 = "Rappen";          // Ein paar Texte
61       printf("S 3 %% von %3d %s und %2d %s%n", z1, t1, z3, t2);
62       printf("T 5 %% von %3d %s und %2d %s%n", z2, t1, z4, t2);
63       printf("-----------------------------------%n");
64       printf("Wiwe01_printfGanz: Das war's erstmal!%n");
65    } // main
```

```
 66    // ------------------------------------------------------------------------
 67 } // class Wiwe01_printfGanz
 68 /* ------------------------------------------------------------------------
 69 Ausgabe:
 70
 71 Wiwe01_printfGanz: Jetzt geht es los!
 72 -----------------------------------
 73 A v01:                 127
 74 B v02:               32767
 75 C v03:          2147483647
 76 D v04: 9223372036854775807
 77 -----------------------------------
 78 E v01: 0000000000000000127
 79 F v02: 0000000000000032767
 80 G v03: 0000000002147483647
 81 H v04: 9223372036854775807
 82 -----------------------------------
 83 I v01:                       127
 84 J v02:                    32.767
 85 K v03:             2.147.483.647
 86 L v04: 9.223.372.036.854.775.807
 87 -----------------------------------
 88 M +100: 100
 89 N -200: -200
 90 -----------------------------------
 91 O +100: +100
 92 P -200: -200
 93 -----------------------------------
 94 Q    12 Euro und 50 Cent
 95 R   520 Euro und 77 Cent
 96 -----------------------------------
 97 S 3 % von  12 Franken und 50 Rappen
 98 T 5 % von 520 Franken und 77 Rappen
 99 -----------------------------------
100 Wiwe01_printfGanz: Das war's erstmal!
101 ------------------------------------------------------------------------ */
```

1. In Zeile A wurde der Wert der Variablen `v01` "auf eine Mindestbreite von 19 Zeichen gepolstert" ausgegeben. Wie viele "Polsterzeichen" wurden dazu vor der Zahl `127` eingefügt?

2. In Zeil E wurde der Wert der Variablen `v01` nochmal ausgegeben.
Mit welchem Zeichen wurde gepolstert?

3. Was ist in den Zeilen E bis H anders als in den Zeilen A bis D?

4. Kommt Ihnen die Zahl in Zeile D (die mit `92...` beginnt) bekannt vor?
Kommt Ihnen die Zahl in Zeile C (die mit `2147...` beginnt) bekannt vor?

5. Was ist in den Zeilen M bis N gleich wie bzw. anders als in den Zeilen Q bis P?

6. In den Zeilen I bis L wurden die Ziffern der Zahlen "landesüblich gruppiert" (in Deutschland mit Punkten. In England/USA wären sie mit Kommas, in Italien mit Blanks, ... gruppiert worden).

Das Projekt 1 bearbeiten:

Die (vollständig vorgegebene) Klasse `AbstractLongSpeicher` erstellen.
Die (unvollständig vorgegebene) Klasse `LongSpeicher10` erstellen.
Die Fragen (auf S. 5 der Datei **Alg-Projekte.odt**) beantworten.
Die Klasse `LongSpeicher10` vervollständigen (siehe S. 6 der Datei **Alg-Projekte.odt**).

SU-02, Do 19.10.2017, Block 2

Heute schreiben wir den Test01.

Kurze Wiederholung

Geben Sie den Namen eines Java-Befehls an, mit dem man einen Wert *ausgeben* kann. (z.B. `printf`)
Geben Sie den Namen des Java-Befehls an, mit dem man einen Wert *zurückgeben* kann. (`return`)
Wo ("in welcher Umgebung") darf man den Befehl `return 123;` benutzen? (In einer Funktion mit dem Ergebnistyp [oder: Rückgabe-Typ, engl. return type] `int`).
Welches Wort kann (und sollte man wohl) statt "zurückgeben" verwenden? (liefern, engl. yield).
Woran erkennt man, ob eine Methode ein *Funktion* ist oder eine *Prozedur*?

if-Anweisungen programmieren

Eine `if`-Anweisung unterscheidet 2 Fälle: Den **dann-Fall** (engl. then case, wenn die nach `if` angegebene Bedingung *erfüllt* ist) und den **sonst-Fall** (engl. else case, wenn die Bedingung *nicht erfüllt* ist).

In vielen Fällen kann man leicht feststellen, dass einer der beiden zu unterscheidenden Fälle deutlich *einfacher* (kürzer, kleiner) ist als der andere. In diesen Fällen ist es meistens empfehlenswert, den *einfacheren Fall* zum *dann-Fall* zu machen (und den *schwierigeren* Fall zum sonst-Fall).
Siehe dazu auch die **Regel-03** auf S. 2 des Papiers **Alg-EinfachSparsamSchoen.odt**.
Wenn man eine `if`-Anweisung fertig geschrieben hat und dann merkt, dass der sonst-Fall viel einfacher ist als der dann-Fall, sollte man erwägen, den `if`-Befehl "umzudrehen" (die Bedingung negieren und die beiden Fälle vertauschen).

Methoden, Parameter und Argumente

Was kann ein ProgrammiererIn mit einer Methode machen? Genau 2 Aktionen sind möglich:
1. Sie kann sie *vereinbaren* (she can *declare* it). Wie oft? (Genau 1 Mal)
2. Sie kann sie *aufrufen* (she can *call* it). Wie oft? (Beliebig oft: 0 Mal, 1 Mal, 2 Mal, ... 17 Mal, ...)

In der *Vereinbarung* einer Methode legt man (0 oder mehr) *Parameter* fest.
In einem *Aufruf* der Methode muss man für jeden Parameter ein entsprechendes *Argument* angeben.
Ein Parameter ist eine spezielle Art von *Variable*.
Als Argument ist ein beliebiger *Ausdruck* (des betreffenden Typs) erlaubt
Beispiele für Ausdrücke vom Typ `int` (unter der Voraussetzung, dass `n` und `m` als `int`-Variablen vereinbart wurden): `17, 2*10, n, n+1, n*m, 2*m+27, Math.max(n, m), ...`

Siehe dazu auch das Papier **Alg-ParameterUebergabe.pdf**, S. 1.

Rekursive Funktionen mit Hilfe von Tabellen verstehen und prüfen

Angenommen, wir wollen von der Fakultätsfunktion `faku` prüfen, ob sie wirklich für alle möglichen Argumente `n` das richtige Ergebnis liefert. Hier nochmal die Vereinbarung der Funktion:

```
1    static int faku(int n) {
2       if (n<=1) return 1;
3       return n * faku(n-1);
4    }
```

Die Prüfung können wir mit einer Tabelle wie der folgenden durchführen:

faku(-1)	faku(0)	faku(1)	faku(2)	faku(3)	faku(4)	faku(5)	...
1	1	1	2	6			

In die obere Zeile schreiben wir Aufrufe der Funktion mit verschiedenen Argumente -1, 0, 1, 2, 3, 4, 5, ... und in die untere Zeile die entsprechenden Funktionswerte. Solang die Argumente kleiner/gleich 1 sind, ist das Funktionsergebnis offenbar immer gleich 1.

Sobald das Argument größer als 1 ist, müssen wir die Zeile 3 ausführen und dabei unter anderem den Wert des rekursiven Aufrufs `faku(n-1)` berechnen. Jetzt kommt der Trick: Diesen Wert haben wir

schon mal berechnet und können ihn jetzt einfach in der Tabelle nachsehen (ohne komplizierte rekursive Aufrufe auszuführen).

Wenn man diesen Trick ein paar Mal angewendet hat, kann man die Tabelle noch ein bisschen vereinfachen und die vielen Wiederholungen von "`faku`" weglassen, etwa so:

n	-1	0	1	2	3	4	5	...
faku(n)	1	1	1	2	6	24	120	

Was für Dinge können rekursiv sein?

Nicht nur *Unterprogramme* (Methoden, Funktionen, Prozeduren) können rekursiv sein, sondern auch *Definitionen* (oder *Beschreibungen*).

Beispiel-01: Wir beschreiben eine eiserne Kette

Beschreibung 1: Eine Eisenkette besteht aus mehreren miteinander verbundenen Kettengliedern.

Diese Beschreibung entspricht (zumindest entfernt) einer Schleife in einem Programm. Es geht aber auch anders:

Beschreibung 2: Eine Eisenkette besteht aus einem Kettenglied, welches mit einer Eisenkette verbunden ist.

Dies ist eine rekursive Beschreibung. Sie ist aber noch nicht ganz richtig. Was fehlt?

Beschreibung 3: Eine Eisenkette ist entweder eine leere Kette, oder
sie besteht aus einem Kettenglied, welches mit einer Eisenkette verbunden ist.

Dies ist eine rekursive Beschreibung mit einem *einfachen Fall* und einem *rekursiven Fall*.

Weitere rekursive Definitionen

Def. Binärzahl: Eine Binärzahl ist
eine Binärziffer
oder eine Binärziffer gefolgt von einer Binärzahl.

Def. Zeichenfolge: Eine Zeichenfolge ist
eine leere-Folge
oder ein Zeichen gefolgt von einer Zeichenfolge.

Def.: Fakultätsfunktion `faku`:

```
faku(0) = 1
faku(n) = n * faku(n-1)  // für alle n>0
```

Def.: Fibonacci-Funktion `fibo`:

```
fibo(0) = 0
fibo(1) = 1
fibo(n) = fibo(n-1) + fibo(n-2) // für alle n>1
```

Aufgabe: Schreiben Sie eine Methode entsprechend der folgenden Spezifikation:

```
static long fibo(int n) {
   // Verlaesst sich darauf, dass n nicht negativ ist.
   // Liefert die n-te Fibonacci-Zahl

   // Lösung der Aufgabe:
   if (n==0) return 0;
   if (n==1) return 1;
   return fibo(n-1) + fibo(n-2);
}
```

Öffnen Sie die Datei **RekursionAufgaben.odt** und betrachten Sie den Anfang der **S. 2**.
Dort soll gezeigt werden, dass die *rekursive Definition* der Fibonacci-Zahlen viel leichter verstehbar ist als die *nicht-rekursive Definition*.

Papier **Alg-RekursionAufgaben.odt**, S. 2, **Aufgabe-12** (Methode `anzWeisen12`)

S. 3, **Aufgabe-13** (Methode `anzWeisen13`).

Zur Entspannung: **Was kostet ein Studium an der Beuth Hochschule?**

Haushalt der Beuth Hochschule ca. 60 Millionen [Euro pro Jahr].
An der Beuth Hochschule studieren ca. 10 Tausend StudentInnen.
Also kostet das Studieren an der Beuth Hochschule ca. 6000 [Euro pro StudentIn und Jahr].
Ein Bachelor-Studium (6 Semester, 3 Jahre) kostet also ca. 18 Tausend [Euros pro StudentIn].

Ü-03, Do 26.10.2017, Block 1, 3, 4

```
 1 // Datei Wiwe02_printfBruch.java
 2 /* ------------------------------------------------------------------------
 3 In diesem Programm werden Bruchzahlen verschiedener Typen mit dem printf-
 4 Befehl auf unterschiedliche Weisen formatiert und zum Bildschirm ausgegeben.
 5 ------------------------------------------------------------------------ */
 6 public class Wiwe02_printfBruch {
 7    // ---------------------------------------------------------------------
 8    static public void main(String[] sonja) {
 9       printf("Wiwe02_printfBruch: Jetzt geht es los!%n");
10       printf("-------------------------------------%n");
11       float  v1 = 12_345_678_901_234.567F;
12       double v2 = 12_345_678_901_234.567D;  // v2 ist etwas kleiner als v1!
13
14       // Standardmaessig mit 6 Nachpunktstellen:
15       printf("A v1: %f%n", v1);
16       printf("B v2: %f%n", v2);
17       printf("-------------------------------------%n");
18       // Mindestbreite 17, davon 2 Nachpunktstellen,
19       // Vorpunktstellen gruppiert
20       printf("C v1: %,17.2f%n", v1);
21       printf("D v2: %,17.2f%n", v2);
22       printf("-------------------------------------%n");
23       // Negative Zahlen beginnen mit '-', positive haben kein Vorzeichen:
24       printf("E +2.5: %.3f%n", +2.5);
25       printf("F -2.5: %.3f%n", -2.5);
26       printf("-------------------------------------%n");
27       // Negative Zahlen beginnen mit '-', positive mit '+':
28       printf("G +2.5: %+.3f%n", +2.5);
29       printf("H -2.5: %+.3f%n", -2.5);
30       printf("-------------------------------------%n");
31       // Mindestbreite 4, davon 1 Nachpunktstelle
32       printf("I +2.5: %+4.1f%n", +2.5);
33       printf("J -2.5: %+4.1f%n", -2.5);
34       printf("-------------------------------------%n");
35       // Betragsmaessig sehr grosse und sehr kleine Zahlen mit Exponent:
36       printf("K         0.000012: %g%n",         0.000012);
37       printf("L 2_000_000.0     : %g%n", 2_000_000.0);
38       printf("-------------------------------------%n");
39       // Langen und kurze Texte kombiniert mit langen und kurzen Zahlen
40       // gut lesbar ausgeben:
41       double d1=1_234_567_890.12, d2 = 43.21;
42       String t1 = "Kanadische Dollar", t2 = "Euro";
43       // Abstand zwischen Text und Zahlen manchmal sehr gross:
44       printf("M %17s: %,16.2f%n", t1, d1);
45       printf("N %17s: %,16.2f%n", t2, d2);
46       printf("-------------------------------------%n");
47       // Abstand zwischen Text und Zahlen manchmal sehr gross:
48       printf("O %,16.2f %17s%n", d1, t1);
49       printf("P %,16.2f %17s%n", d2, t2);
50       printf("-------------------------------------%n");
51       // Abstand zwischen Text und Zahlen immer klein, weil der Text jetzt
52       // linksbuendig formatiert wird (wegen Minuszeichen -).
53       printf("Q %,16.2f %-17s%n", d1, t1);
54       printf("R %,16.2f %-17s%n", d2, t2);
55       printf("-------------------------------------%n");
56       printf("Wiwe02_printfBruch: Das war's erstmal!%n");
57    } // main
58    // ---------------------------------------------------------------------
59    // Eine Methode mit einem kurzen Namen:
60    static void printf(String f, Object... v) {System.out.printf(f, v);}
61    // ---------------------------------------------------------------------
62 } // class Wiwe02_printfBruch
63
```

```
64
65 /* ------------------------------------------------------------------------
66 Ausgabe:
67
68 Wiwe02_printfBruch: Jetzt geht es los!
69 ---------------------------------------
70 A v1: 12345679020032,000000
71 B v2: 12345678901234,566000
72 ---------------------------------------
73 C v1: 12.345.679.020.032,00
74 D v2: 12.345.678.901.234,57
75 ---------------------------------------
76 E +2.5: 2,500
77 F -2.5: -2,500
78 ---------------------------------------
79 G +2.5: +2,500
80 H -2.5: -2,500
81 ---------------------------------------
82 I +2.5: +2,5
83 J -2.5: -2,5
84 ---------------------------------------
85 K         0.000012: 1,20000e-05
86 L 2_000_000.0     : 2,00000e+06
87 ---------------------------------------
88 M Kanadische Dollar: 1.234.567.890,12
89 N              Euro:            43,21
90 ---------------------------------------
91 O 1.234.567.890,12 Kanadische Dollar
92 P            43,21              Euro
93 ---------------------------------------
94 Q 1.234.567.890,12 Kanadische Dollar
95 R            43,21 Euro
96 ---------------------------------------
97 Wiwe02_printfBruch: Das war's erstmal!
```

1. Wie viele Bits belegt ein `float`-Wert? Und ein `double`-Wert?

2. Ungefähr wie groß ist der Wert von `v1` (auf 2 Ziffern genau)?

3. Ungefähr um wie viel ist der Wert von `v1` kleiner als der von `v2`?

4. Was bedeutet in Zeile 20 (im Formatstring des `printf`-Befehls) die Angabe `17.2`?
Der `double`-Wert `17.2` soll ausgegeben werden?
`17` Stellen *vor* und `2` Stellen *nach* dem Dezimal-Trennzeichen? Oder was?

5. Wodurch unterscheiden sich die `printf`-Befehle in den Zeilen 28 und 29 von den `printf`-Befehlen in den Zeilen 24 und 25?

6. Was bewirkt dieser Schalter `+` (in den Zeilen 28 und 29)?

7. In Zeile 85 und 86 sieht man Bruchzahlen, die mit `%g` (statt mit `%f`) formatiert wurden.
Dieses Format für Bruchzahlen heißt "wissenschaftliche Notation" oder "Exponentialschreibweise".

8. In den Zeilen 88-89 bzw. 91-92 sind manchmal große Zwischenräume zwischen Zahlen und Texten. In den Zeilen 94-95 gibt es keine solchen Zwischenräume. Was bewirkt das Minuszeichen im Umwandlungsbefehl `%-17s` in Zeilen 53 und in Zeile 54?

Hinweis: Die Schalter `+` und `-` beeinflussen ganz unterschiedliche Dinge:
Das Pluszeichen beeinflusst die Ausgabe von positiven Zahlen
Das Minuszeichen bewirkt eine linksbündige Ausgabe.

Antworten: (32 bzw. 64 Bits), (12 Billionen), (Ungefähr um 100 Tausend, oder etwas genauer, aber auch nicht ganz genau: Um `118.798`), (Mindestbreite 17, davon 2 nach dem Dezimal-Trennzeichen), (Durch ein `+`-Zeichen nach dem `%`-Zeichen), (Positive Zahlen beginnen mit einem Pluszeichen, negative Zahlen mit einem Minuszeichen, siehe Zeilen 79 und 80), (), (Der String wird *linksbündig* ausgegeben).

SU-03, Do 26.10.2017, Block 2

Heute schreiben wir den Test02.

Kurze Wiederholung

Was kann ein Programmierer mit einer Methode machen? (vereinbaren, aufrufen)
Wo kommen Parameter vor? (in Methoden-Vereinbarungen)
Wo kommen Argumente vor? (in Methoden-Aufrufen)
Woran sollte man beim Schreiben von if-Anweisungen denken (einfachen Fall als dann-Fall)

Anrede in Emails

Einfache Anreden (z.B. "Hallo Herr Grude") sind besser als komplizierte Anreden mit Titeln.

Rekursion beim Treppen-Steigen

Papier **Alg-RekursionAufgaben.odt**, S. 2, **Aufgabe-12** (und evtl. S. 3, **Aufgabe-13**)

Diese Aufgaben sollen verdeutlichen: Das *Finden* einer rekursiven Lösung kann ziemlich *schwierig* sein. Aber wenn man die Lösung dann gefunden hat, ist sie oft erstaunlich *kurz und einfach*. Und wenn man sich die Lösung merkt, ist man in Zukunft schlauer :-).

Abstrakte Zahlen und konkrete Darstellungen von Zahlen

Etwa im Jahr 1200 fingen auch Europäer an, Zahlen nicht mehr als römische Zahlen darzustellen, sondern im *Dezimalsystem*. Sie lernten das von arabischen Wissenschaftlern, und die hatten es von indischen Wissenschaftlern übernommen.

Kennen Sie ähnliche Zahlensysteme wie das *Dezimalsystem*? *Wie viele* solcher Systeme gibt es?

In dieser LV werden wir Zahlen-Systeme wie folgt bezeichnen: 1-er-System, 2-er-System. 3-er-System, ..., 10-er-System, 11-er-System, ..., 137-er-System, ..., 4_294_967_296-er-System, ...

Diese Systeme heißen *Stellenwertsysteme* (oder: *Positionssysteme*), weil der Begriffe **"Stelle"** und **"Stellenwert"** darin ein wichtige Rolle Spielen.

Beispiel-01: Die folgende Tabelle enthält die 10-er-Zahl **2375** sowie
die Stellen, Stellenwerte und Beiträge der einzelnen Ziffern:

Stellen	...	3	2	1	0
Stellenwerte (Formeln)	...	10^3	10^2	10^1	10^0
Stellenwerte (ausgerechnet)	...	1000	100	10	1
Ziffern einer Zahl	...	**2**	**3**	**7**	**5**
Beiträge der Ziffern	...	2000	300	70	5

Wo wird das 1-er-System häufig benutzt? (Auf Bierdeckeln)
Wo wird das 4_294_967_296-er-System benutzt?
(Beim Darstellen und Rechnen mit sehr großen Zahlen, z.B. in der Klasse `BigInteger`)

Wie viele Ziffern hat die Zahl hundert? Drei Ziffern? Stimmt das immer? Die Darstellung 1100100_2 der Zahl hundert hat 7 Ziffern, und andere Darstellungen von hundert haben mehr oder weniger Ziffern.
Moral: Man muss zwischen abstrakten Zahlen und konkreten Darstellungen (von Zahlen) unterscheiden.

Welche der folgenden Eigenschaften E1 bis E7 betreffen *abstrakte Zahlen* (AZ) und welche betreffen *konkrete Darstellungen* (KD) von Zahlen?

E1: Ist ungerade	AZ
E2: Besteht aus 3 Ziffern	KD
E3: Ist prim	AZ
E4: Endet mit der Ziffer 7	KD
E5: Ist größer als drei	AZ
E6: Hat die Quersumme acht	KD
E7: Kann in Java als `int`-Wert dargestellt werden	AZ

In welchem System werden int-Werte dargestellt?
`int`-Werte sollte man als *abstrakte Zahlen* betrachten, die nicht in einem bestimmten Zahlensystem dargestellt sind, die man aber in *allen möglichen Zahlensystemen* darstellen kann, wenn man das möchte.

Angenommen, wir haben (in einem Java-Programm) eine `int`-Variable `n`, deren Wert nicht negativ ist. Wir wollen den Wert von `n` z.B. als 10-er-Zahl (oder als 17-er-Zahl, oder als ...) darstellen. Wie können wir die einzelnen Ziffern dieser Darstellung berechnen?

Die Ziffer (einer nicht-negativen Ganzzahl n) an der Stelle 0 berechnen:
Der Ausdruck `n%10` liefert die Ziffer an der Stelle 0 der Darstellung von `n` im 10-er-System
Der Ausdruck `n%17` liefert die Ziffer an der Stelle 0 der Darstellung von `n` im 17-er-System
Der Ausdruck ...

Die entsprechenden Ausdrücke für Zahlen im 2-er-System oder im 137-er-System oder ... sollten Sie selbst herausfinden.

Was ist `5 / 2`? Die Antwort "`2,5`" ist manchmal richtig, aber manchmal ist sie falsch.
Wenn ein Java-Ausführer den Wert des Ausdrucks `5 / 2` berechnet, ist das Ergebnis gleich `2`
(Literale wie `2`, `3`, `5`, `123` etc. sind in einem Java-Programm immer vom Typ `int`, und wenn der Ausführer mit `int`-Werten rechnet, ist das Ergebnis immer ein `int`-Wert, d.h. eine Ganzzahl, keine Bruchzahl).

Wie kann man die Ziffern an den anderen Stellen (Stelle 1, 2, 3 etc.) berechnen?
Indem man sie an die Stelle 0 verschiebt.
Wie kann man Ziffern innerhalb einer Zahl verschieben?
Indem man die Ziffer an der Stelle 0 entfernt (dadurch rutschen alle übrigen Ziffern eine Stelle nach rechts).

Die Ziffer an der Stelle 0 entfernen:
Der Ausdruck `n/10` bezeichnet "die Zahl `n` ohne die Ziffer an der Stelle 0 im 10-er-System"
Der Ausdruck `n/17` bezeichnet "die Zahl `n` ohne die Ziffer an der Stelle 0 im 17-er-System"
Der Ausdruck ...

Beispiele:

`123 / 10` ist gleich `12`,
`1234 / 10` ist gleich `123`,
`72387 / 10` ist gleich `7238`

Papier **Alg-RekursionAufgaben.odt**, S. 5, **Aufgabe-211**, **Aufgabe-21**2, **Aufgabe-213**

Zur Entspannung: **Hilberts Hotel**

Denken Sie sich ein Hotel mit unendlich vielen, nummerierten Zimmern: 1, 2, 3, ... Alle Zimmer sind belegt. Dieses Hotel wurde nach dem Mathematiker David Hilbert (1862-1943) benannt.

1. Wie kann man *einen* weiteren Gast unterbringen? (`ZrNr := ZrNr + 1`)
2. Wie kann man *hundert* weitere Gäste unterbringen? (`ZrNr := ZrNr + 100`)
3. Wie kann man *unendlich* viele weitere Gäste unterbringen? (`ZrNr := ZrNr * 2`)
danach sind alle Zimmer mit ungeraden Nrn. (1, 3, 5, ...) frei.

Ü-04, Do 02.11.2017, Block 1, 3, 4

Intervall-Prüfungen

Schreiben Sie eine Methode entsprechend der folgenden Spezifikation:

```
1    static boolean innerhalb(int n, int von, int bis) {
2       // Liefert true, wenn n innerhalb des Intervalls zwischen
3       // von und bis (einschliesslich) liegt, und liefert sonst false.
```

Für diese Aufgabe gibt es sehr viele korrekte und ähnlich gute Lösungen. Es wäre aber gut, wenn man die Aufgabe möglichst immer "auf die gleiche Weise" löst, und in einem großen Programm nicht viele verschiedene Lösungen für die gleiche Aufgabe vorkommen.

Öffnen Sie die Datei **Alg-EinfachSparsamSchoen.odt** auf **S. 4**.

Betrachten Sie (oben im Kasten) die **Regel-06a** und die **Regel-06b**.

Ich hoffe dass ich Sie davon überzeugen kann, dass diese Regeln zu Lösungen führen, die ein bisschen schöner sind als alternative Lösungen (aber möglicherweise finden Sie eine andere Lösung noch schöner).

Innerhalb einer Reihung eine Teilreihung verschieben

Wenn man Methoden programmiert, die Reihungen (engl. arrays) bearbeiten, ist es häufig günstig, Beispiele für Reihungen irgendwie grafisch darzustellen.

Eine Reihung `speicher` von 7 `long`-Variablen (und die Variable `lbi`) kann man z.B. so darstellen:

0	1	2	3	4	5	6
20	30	40	60	70	80	0

lbi: 5

Angenommen, wir wollen beim Index 3 (wo jetzt noch 60 steht) den `long`-Wert 50 einfügen, nachdem wir alle Komponenten mit den Indizes 3 bis 5 *um eine Position nach rechts* verschoben haben.
Was würde passieren, wenn wir dabei *von links nach rechts* vorgehen würden?
Also müssen wir *von rechts nach links* vorgehen.

Angenommen, wir wollen die Komponente 40 (die mit dem Index 2) löschen, indem wir alle Komponenten mit den Indizes 3 bis 5 um eine Position nach nach links verschieben.
Was würde passieren, wenn wir dabei *von rechts nach links* vorgehen würden?
Also müssen wir *von links nach rechts* vorgehen.

Öffnen Sie die Datei **Alg-EinfachSparsamSchoen.odt** auf **S. 5**

Dort werden kanonische Lösungen für solche Verschiebe-Probleme vorgeschlagen.

SU-04, Do 02.11.2017, Block 2

Heute schreiben wir den Test 3

Kurze Wiederholung

Wie viele Stellenwertsysteme (zur Darstellung von Zahlen) gibt es? (Abzählbar unendlich viele)
An welcher Stelle steht die Ziffer `B` in der Zahl `3B67A95`? (An der Stelle 5)
Welcher Stellenwert ist der Stelle 3 im 10-er-System zugeordnet? (10^3 gleich 1000)
Welcher Stellenwert ist der Stelle 3 im 2-er-System zugeordnet? (2^3 gleich 8)
Welcher Stellenwert ist der Stelle 3 im 5-er-System zugeordnet? (5^3 gleich 125)

Wozu sind Logarithmen gut?

Abkürzung: "10-er-Logarithmus" soll "Logarithmus zur Basis 10" bedeuten.
Entsprechend auch 2-er-Logarithmus, 17-Logarithmus etc.

Warum ist der 10-er-Logarithmus von 100 gleich 2? (Weil 10^2 gleich 100 ist)
Warum ist der 2-er-Logarithmus von 8 gleich 3? (Weil 2^3 gleich 8 ist)

Was ist der 10-er-Logarithmus von 1000? (3)
Was ist der 2-er-Logarithmus von 32? (5)

Bevor es Taschenrechner gab war Folgendes sehr wichtig: Mit Hilfe von Logarithmentafeln kann man (aufwendige) *Multiplikationen* durch (deutlich weniger aufwendige) *Additionen* ersetzen. Das war unter anderem bei der Navigation von Schiffen auf hoher See wichtig.

Heute sind Logarithmen manchmal beim Programmieren nützlich:

Wie viele Ziffern braucht man, um eine (positive) Zahl z als 10-er-Zahl darzustellen?
(etwa $\log_{10}(z)$ viele Ziffern).

Wie viele Ziffern braucht man, um eine (positive) Zahl z als b-er-Zahl darzustellen?
(etwa $\log_b(z)$ viele Ziffern).

Wie oft kann/muss man eine Reihung (engl.: an array) der Länge n in zwei (ungefähr) gleich lange Hälften teilen, und diese Hälften in Viertel teilen etc., bis man lauter Reihungen der Länge 1 hat?
(etwa $\log_2(n)$ mal).

Im Fach Algorithmen ist der Logarithmus zur Basis 2 besonders wichtig.
Mit **log(n)** ist ab jetzt immer **$\log_2$(n)** ("der Logarithmus zur Basis 2") gemeint.

Was ist so toll an sortierten Reihungen?

Vorteile von Reihungen: Ein Zugriff auf eine Reihungskomponente `r[i]` braucht auf heute üblichen Rechnern sehr wenig Zeit und diese Zeit ist unabhängig von der Größe des Index `i`.
Nachteil: Die Länge `r.length` einer Reihung ist unveränderbar ("Reihungen sind aus Beton").

In einer *sortierten Reihung* kann man *sehr schnell* suchen (z.B. mit der Methode `istDrin`). Der entsprechende Such-Algorithmus wird *binäres Suchen* genannt.

Beispiel: Angenommen, wir haben eine sortierte Reihung von `long`-Werten namens `r`:

```
long[] r = {8, 13, 21, 35, 47, 59, 62, 74, 86, 97, 108, 114, 129, 134, 145};
```

Index i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
r[i]	**8**	**13**	**21**	**35**	**47**	**59**	**62**	**74**	**86**	**97**	**108**	**114**	**129**	**134**	**145**
Schritt	4	3	4	2	4	3	4	1	4	3	4	2	4	3	4

Angenommen, wir suchen die Zahl n gleich 62.
Dann vergleichen wir n der Reihe nach mit
`r[7]` (n ist kleiner), `r[3]` (n ist größer), `r[5]` (n ist größer), `r[6]` (n ist gleich, gefunden)

Angenommen, wir suchen die Zahl n gleich 108.
Dann vergleichen wir n der Reihe nach mit
`r[7]` (n ist größer), `r[11]` (n ist kleiner), `r[9]` (n ist größer), `r[10]` (n ist gleich, gefunden)

Angenommen, wir suchen die Zahl n gleich 29.
Dann vergleichen wir n der Reihe nach mit
`r[7]` (n ist kleiner), `r[3]` (n ist kleiner), `r[1]` (n ist größer), `r[2]` (n ist ungleich, nicht gefunden)

Der folgende Baum soll deutlich machen, bei welchem Schritt wir `n`
mit welcher Komponenten `r[i]` vergleichen (angegeben ist nur der Index `i`):

```
Schritt 1:                    7
                    +---------+----------+
Schritt 2:          3                    11
                +---+---+             +---+---+
Schritt 3:      1       5             9       13
              +-+-+   +-+-+         +-+-+   +-+-+
Schritt 4:    0   2   4   6         8   10  12  14
```

Im **Projekt 2** (genau wie im im **Projekt 1**) wollen wir nicht "in der gesamten Reihung namens `speicher`" suchen, sondern nur im "schon belegten Teil" `speicher[0 .. lbi]`.

Die (ungefähre) **Mitte einer Teilreihung** berechnen: Angenommen, `von` und `bis` sind zwei Indizes für die Reihung `speicher` (und `von` ist nicht größer als `bis`). Wie kann man den Index berechnen, der möglichst genau zwischen `von` und `bis` liegt?

```
int mitteF = (von + bis) / 2;     // FALSCH Ueberlauf fuer sehr grosse von und bis
int mitteR = von + (bis-von) / 2; // RICHTIG
```

Beispiel: Wenn `von` gleich 1 Milliarde und `bis` gleich 1.5 Milliarden ist, dann ist

```
mitteF:  -897.483.648             // FALSCH
mitteR: 1.250.000.000             // RICHTIG
```

Programmieren Sie: Papier **Projekte.odt**, S. 8, die Methode `int index(long n)` **iterativ**
(d.h. "mit einer Schleife", nicht rekursiv)
Setzten Sie dabei voraus, dass die Teilreihung `speicher[0 .. lbi]` *aufsteigend sortiert* ist.

Programmieren Sie: Papier **Projekte.odt**, S. 9, die Methode `int indexR(long n)` **rekursiv**.

Bearbeiten Sie (im Papier **Projekte.odt** auf **S. 10**) die **Aufgabe-01**

Angenommen, die Variable `lbi` (letzter belegter Index) hat den Wert 4 und die Reihung (the arrya) `speicher` enthält 5 `long`-Werte wie folgt:

Die Reihung `speicher`:

Indizes	0	1	2	3	4
long-Werte	20	30	40	50	60

Berechnen Sie mit Papier und Bleistift die Ergebnisse der Funktion `index` für verschiedene Parameter und tragen Sie Ihre Ergebnisse in die folgende Tabelle ein:

Lösung für Aufgabe-01: Ergebnisse der Funktion `index`:

n	10	20	25	35	55	65
index(n)	**0**	**0**	**1**	**2**	**4**	**5**

Zur Entspannung: **Al-Khwarizmi (ca. 780-850)**

Abu Ja'far Muhammad ibn Musa Al-Khwarizmi war ein islamischer Mathematiker, der in Bagdad lebte, über Indisch-Arabische Zahlen schrieb und zu den ersten gehörte, die die Ziffer 0 benutzten. Aus seinem Namen Khwarizmi entstand (durch Übertragung ins Latein und dann in andere Sprachen) das Wort **Algorithmus**. Eine seiner Abhandlungen heißt *Hisab al-jabr w'al-muqabala* (auf Deutsch etwa: "Über das Hinüberbringen". Gemeint ist damit: Von einer Seite einer Gleichung auf die andere). Aus dem arabischen *al-jabr* entstand unser Wort **Algebra**.

Ü-05, Do 09.11.2017, Block 1, 3, 4

Programmieren als Rollenspiel (siehe auch S. 1 des Buches)

Man kann die Tätigkeit des Programmierens als eine Art Rollenspiel (ähnlich einem Theaterstück) mit folgenden Rollen beschreiben:

Rolle	**Tätigkeiten**
Programmierer	*Schreibt* Programm, *übergibt* sie dem Ausführer.
Benutzer	*Befiehlt* dem Ausführer, Programme *auszuführen, ist für E/A-Daten zuständig*.
Ausführer	*Prüft* Programme (wenn sie ihm übergeben werden), *akzeptiert* sie/*lehnt* sie *ab*, *führt* Programme *aus* (wenn der Benutzer es ihm befiehlt).
Warter	*Wartet* Programme (d.h. korrigiert, erweitert, verändert sie).
Wiederverwender	Versucht, möglichst viele Programmteile *wiederzuverwenden*.

Warter und Wiederverwender bezeichnen wir zusammen auch als die **Kollegen** (des Programmierers).

Es folgen hier ein paar mögliche **Besetzungen** der oben beschriebenen Rollen.
Dabei bezeichnet **Priscilla** eine Programmiererin, die z.B. Java-Programme schreiben kann.

Programmierer	**Benutzer**	**Ausführer**	**Kollegen**
Priscilla	Priscilla	ein Computer	Priscilla
ein Team	ein Team	ein Computer	ein Team
Priscilla	Priscilla	Priscilla	Priscilla

Mit Priscilla ist ein Mensch gemeint, der im Prinzip alle Rollen übernehmen kann. Priscilla sollte Ihr Vorbild sein (insbesondere sollten Sie jeden Befehl, den Sie als Programmierer *schreiben*, im Prinzip auch selbst *ausführen* können).

Def.: Ein Programm ist eine Folge von Befehlen, die von einem *Programmierer* geschrieben wurde und von einem *Ausführer* ausgeführt werden kann.

Projekt 2 bearbeiten und vorführen.

SU-05, Do 09.11.2017, Block 2

Heute schreiben wir den Test04

Java-Befehle ins Deutsche übersetzen

Um Java-Befehle besser zu verstehen, kann man sie ins Deutsche übersetzen.

Besipiel-01: Eine Variablenvereinbarung

```
int otto = 2*emil;
```

Auf Deutsch: Erzeuge eine Variable namens `otto` vom Typ `int` mit dem Anfangswert `2*emil`.

Beispiel-02: Eine Methoden-Vereinbarung

```
static public void ph() {System.out.printf("Hallo!");}
```

Auf Deutsch: Erzeuge eine öffentliche Klassenmethode namens `ph`, mit dem Rückgabetyp (oder: Ergebnistyp) `void` und 0 Parametern, die den String `"Hallo!"` zur Standardausgabe ausgibt.

Bojen (engl: buoys)

In Java unterscheidet man 2 Gruppen von Typen. Wie heißen die? (primitive Typen, Referenztpyen).

Namen aller primitiven Typen? (byte, short, char, int, long, float, double, boolean)

Beispiele für Referenztypen? (String, StringBuilder, Integer, Arrays, JButton, int[], String[], ...)

Def.: Ein Typ ist ein Bauplan für Variablen.

Wie stellen Sie sich eine `int`-Variable vor? Aus wie vielen und welchen Teilen besteht sie?
Und eine `String`-Variable?

Öffnen Sie das Papier **Alg-JavaBuoys.pdf** auf S. 2

Hinweise: Wer Mühe hat, englische Texte zu lesen, kann auch im Buch nachlesen und lernen, was Bojen sind. Im Stichwortverzeichnis sind zum Wort "Boje" die Seitenzahlen aller Stellen aufgeführt, an denen etwas über Bojen steht.

Example-01: `anna` ist eine primitive Variable, `bert` und `carl` sind Referenzvariablen.

Welchen Wert hat die Variable `anna`? (17)
Welchen Wert hat die Variable `bert`? (<22>)
Welchen Wert hat die Variable `carl`? (<null>)

Eine Variable besteht aus mindestens **2 Teilen** (Referenz und Wert) und
höchstens aus **4 Teilen** (Name, Referenz, Wert, Zielwert)

Der *Programmierer* hat direkt nur mit den *Namen* von Variablen zu tun. Für die anderen Teile ist der *Ausführer* (engl: exer) zuständig. Die Referenzen von allen Variablen und die Werte von Referenzvariablen werden vom *Ausführer* festgelegt, nicht vom *Programmierer.*

Garantie des Ausführers: Verschiedene Variablen haben verschiedene Referenzen.

Die *Referenz* einer Variable wird auch als "Identität der Variablen" bezeichnet.
Sie hat Ähnlichkeit mit der Personalausweis-Nr eines Menschen.

Eine Referenzvariable hat entweder
einen *normalen Wert* (wie z.B. `bert`) und zeigt (oder: referiert) damit auf *ein Objekt*, oder
sie hat den (*speziellen*) *Wert* `null` (sprich: nall), der auf *kein* Objekt zeigt (wie z.B. `carl`).

Merke: Nur **Referenzvariablen** können den Wert `null` haben (primitive Variablen können das *nicht*!)

Mit der Bojen-Darstellung kann man sich genau klarmachen, was eine Zuweisung bewirkt.

Example-02: Zwei Zuweisungen und was sie bewirken.

Example-03: Was bewirkt ein Aufruf einer `equals`-Methode?

In Java enthält jedes Objekt eine `equals`-Methode (mit einem Parameter vom Typ `Object`).
Was diese Methode genau macht, hängt von der Klasse ab, zu der das Objekt gehört.

Achtung: Ein `String`-Objekt enthält (bei Java 8) 77 Methoden, ein `StringBuilder`-Objekt enthält nur 65 Methoden. Auf jeden Fall ist ein `String`-Objekt etwas ganz anderes als ein `StringBuilder`-Objekt.

Fortsetzung von **Example-03** (auf S. 5): Was bewirkt ein Aufruf der Operation `==` (oder `!=`)?

Spezielle Regeln für den Typ String, Example-05 (S. 5)

Zur Entspannung: **Christian Morgenstern** (1871-1914)

Der Werwolf

Ein toter Lehrer gibt einem Werwolf Nachhilfeunterricht in Grammatik

Ein Werwolf eines Nachts entwich / von Weib und Kind und sich begab
an eines Dorfschullehrers Grab / und bat ihn "Bitte, beuge mich!"

...

Ü-06, Do 16.11.2017, Block 1, 3, 4

Angenommen, wir haben folgende Referenzvariablen:

```
StringBuilder xxx = new StringBuilder("ABC");
String        yyy = new String("DEF");
```

Die vier Teile der Variablen `xxx` und `yyy` als (ASCI-) Bojen dargestellt:

```
Name   Referenz   Wert    Zielwert
|xxx|---<300>----[<310>]---["ABC"]
|yyy|---<400>----[<410>]---["DEF"]
```

In jedem der folgenden 8 Befehle kommt der Name einer Referenzvariablen vor.
Welchen *Teil* der Variable bezeichnet dieser Name in den einzelnen Befehlen?
(Auch hier soll `pln` eine Abkürzung für `System.out.println` sein).

```
                              // Der Name xxx bzw. yyy bezeichnet:
1   if (xxx == ...) ...       // Den Wert [<310>]
2   if (yyy == ...) ...       // Den Wert [<410>]
3   if (xxx.equals(...)) ...  // Den Wert [<310>]
4   if (yyy.equals(...)) ...  // Den Zielwert ["DEF"]
5   xxx.append("ZZ");         // Den Zielwert ["ABC"]
6   xxx = ... ;               // Die Referenz <300>, die auf das Wertkästchen
                              // zeigt (der "alte Wert" [<310>] ist hier
                              // nicht wichtig, er wird ersetzt).
7   ... = xxx;                // Den Wert [<310>]
8   pln(xxx);                 // Den Wert oder den Zielwert, je nachdem
                              // ob der Wert gleich oder ungleich null ist
                              // (ausgegeben wird null bzw. der Zielwert).
```

Man muss also immer aus dem Zusammenhang erkennen, welcher *Teil* der Variablen
mit dem *Namen* der Variablen gemeint ist: die Referenz, der Wert oder der Zielwert.

SU-06, Do 16.1.2017, Block 2

Heute schreiben wir den Test05

Kurze Wiederholung:
Aus wie vielen und welchen Teilen besteht eine Variable mindestens? (2 Teile, Referenz, Wert)
Aus wie vielen und welchen Teilen besteht eine Variable höchstens? (4 Teile, + Name, Zielwert)
Aus wie vielen Teilen besteht eine primitive Variable höchstens? (3 Teile, Name, Referenz, Wert)
Welche Variablen können den Wert `null` haben? (Alle Referenzvariablen)
Was wird ausgegeben durch:

```
String s = "ABC";
s.replace('A', 'B');
println(s)?
```

(Ausgegeben wird `ABC`)
Was ist ein Attribut (engl. a field)?
(Eine Variable, die *direkt* innerhalb einer Klasse vereinbart wurde. Eine Variable, die innerhalb einer Methode innerhalb einer Klasse vereinbart wurde, ist eine *lokale Variable* [der Methode], aber kein Attribut)

Hatte jemand Probleme beim Lösen der Rekursion-Aufgaben?

Emails an mich

Auch an alle Mitglieder der Arbeitsgruppe schicken (damit ich dann "Allen antworten" kann).

Betreff-Zeile: Wenn z.B. die Arbeitsgruppe **Alk03** mir ihre Datei **LongSpeicher30.java** schickt, dann sollte die Betreff-Zeile so aussehen:

`Algorithmen, Alk03, LongSpeicher30`

Ein wichtiger Grundbegriff

Def.: Ein *Wertebehälter* ist entweder eine *Variable* oder ein *E/A-Gerät* wie z.B. eine Tastatur, ein Bildschirm, ein Drucker, eine (Datei auf einer) Festplatte, eine CD etc.

Drei Arten von Befehlen (und ihre Übersetzungen ins Deutsche)

Es gibt kaum überschaubar **viele** Programmiersprachen (tausende! Nicht nur hunderte!) und in jeder Programmiersprache **zahlreiche** Befehle (die der Programmierer dem Ausführer geben kann). Trotzdem kann man alle diese Befehle sinnvoll in nur *drei Gruppen* oder *Arten* einteilen. Und wenn man von einem Befehl nur weiß, zu welcher Gruppe er gehört, dann weiß man schon viel darüber, "was er macht" und wie man ihn ins Deutsche übersetzen kann/sollte.

Befehls-Art	**Beispiele**	**Übersetzung ins Deutsche**

Vereinbarung	`int otto = 17;`
	`int plus1(int n) {return n+1;}`
	`class carl extends otto {...}`
		Erzeuge ... (eine Variable, eine Methode, eine Klasse, ...)

Ausdruck	`otto + 5, plus1(2*otto), 17, otto`	Berechne den Wert des Ausdrucks ...

Anweisung	`otto = otto + 3;`
	`println(otto + " Kilogramm");`
	`return otto%2 != 0;`
		Tue den Wert ... in den Wertebehälter ... (evtl. auch Plural)

Merke: Jeder Aufruf einer *Funktion* ist ein *Ausdruck*, jeder Aufruf einer *Prozedur* eine *Anweisung*.
Jedes *Literal* (z.B. `17, 3.5, 3.5F, 'A', "Hallo", true`) ist ein *Ausdruck*.
Jeder Variablen-Name ist ein *Ausdruck*.
Auf der rechten Seite einer Zuweisung muss ein *Ausdruck* stehen (z.B. `... = otto + 3;`).

Ein *Ausdruck mit Seiteneffekt* ist gleichzeitig ein *Ausdruck* und eine *Anweisung*.

Beispiele für Ausdrücke mit Seiteneffekt:

```
otto++      // Hat den Wert von otto und erhöht otto (dann) um 1
++otto      // Erhöht otto um 1 und hat (dann) den Wert von otto
otto=otto+3 // Ohne Semikolon! Hat den Wert otto+3 und ändert otto zu otto+3
```

Ausdrücke mit Seiteneffekt sollte man vorsichtig einsetzen. Wenn mehrere davon ausgewertet werden, ist ihre Reihenfolge wichtig. Wenn mehrere *normale Ausdrücke* ausgewertet werden, ist die Reihenfolge egal (das erleichtert das Lesen und ermöglicht dem Ausführer bestimmte Optimierungen).

Einfach verkettete Listen

Was kann man innerhalb einer Klassen-Vereinbarung alles vereinbaren?
(Attribute, Methoden, Klassen, Schnittstellen, Konstruktoren).

Schlagen Sie in der Datei **Projekte.odt** die **S. 13** auf und sehen Sie sich die vorgegebene Version der Datei `LongSpeicher30.java` an.

In der Klasse `LongSpeicher30` wird eine Klasse `Knoten` vereinbart (ab Zeile 17).
Die Klasse `Knoten` ist deshalb eine *geschachtelte* Klasse (engl. nested class).
Wie viele und was für Elemente werden in jedes `Knoten`-Objekt eingebaut?
(Nur 2 Attribute namens `next` und `data`)

In der Bojendarstellung stellen wir ein **Objekt** einfach als **rechteckigen Kasten** dar, in den wir die **Attribute** des Objekts einzeichnen (weil die *Werte* der Attribute sich ändern können). Andere Elemente, die sich in dem Objekt befinden (Methoden, Klassen, Schnittstellen) zeichnen wir nicht ein (da sie sich meistens nicht ändern und immer so aussehen, wie in der Vereinbarung der betreffenden Klasse vereinbart).

Anmerkung: In Java sind die *Zielwerte von Referenzvariablen* immer *Objekte* und *Objekte* gibt es nur als *Zielwerte von Referenzvariablen*. In anderen Sprachen kann eine Referenzvariable auch auf primitive Werte zeigen und ein Objekt kann auch als Wert (statt als Zielwert) einer Variablen existieren.

Angenommen, wir vereinbaren irgendwo eine `LongSpeicher30`-Variable wie folgt:

```
1 LongSpeicher30 lob = new LongSpeicher30();
```

Wie sieht diese Variable als Boje dargestellt aus?

Ein Blatt mit einer Boje einer `LongSpeicher30`-Variablen namens `lob` austeilen. Auf diesem Blatt sollen dann mit Bleistift und Radiergummi nacheinander 3 `long`-Werte eingefügt werde. Die Methode `fuegeEin` ist auf dem Blatt vorgegeben wie folgt:

```
public boolean fuegeEin(long n) {
   // Fuegt n in diesen Speicher ein und liefert true.

   // Einen neuen Knoten mit n als data vorn in die Liste einfuegen:
   ADK.next = new Knoten(ADK.next, n);
   return true;
}
```

Zur Entspannung: **Ein paar englische Vokabeln**

ambiguous	zweideutig, unklar
geek	Fachmann (z. B. für Computer), mild negativ. Früher war ein *geek* "ein wilder Mann mit Bart" auf einer Kirmes, der z. B. Mäusen den Kopf abbiss.
to execute	ausführen (z. B. ein Programm oder einen Befehl), hinrichten (z. B. einen Verurteilten, nur in Ländern mit Todesstrafe).
rocket science	wörtlich: Raketenwissenschaft, sonst: schwierig zu lernen, anspruchsvoll.
to drag	ziehen, zerren (z. B. eine Maus über eine Tischplatte).
what a drag	Was für ne Mühe, Umstand.
in full drag	aufgebrezelt, aufgetakelt, auffällig zurecht gemacht.
drag queen	Transvestit.
for the birds	für die Katz (wörtlich: für die Vögel), bringt nichts, unnütz, z. B. im Wortspiel *nesting (of functions) is for the birds*.