Stichworte

zur Lehrveranstaltung **Informatik 3** (TB3-IN3)
im Studiengang **Technische Informatik Bachelor**
im **Wintersemester 2011/12** an der **Beuth-Hochschule für Technik** Berlin
von Ulrich Grude.

Gegenstand dieser Lehrveranstaltung: Objektorientierte Programmierung (mit Java)

In dieser Datei finden Sie nach jedem seminaristischen Unterricht (nach jeder Vorlesung) Stichworte zum Stoff, der im letzten SU behandelt wurde.

Die Regeln, nach denen Sie in diesem Semester eine Note für das Fach TB3-IN3 erwerben können, stehen am Anfang der Datei http://public.beuth-hochschule.de/~grude/AufgabIN3_11WS.pdf.

1. SU Mo 26.09.2011

im Raum B325, 3. Block (12.15 - 13.45 Uhr)

1. Begrüßung

Namensschilder

2. Organisation der LV

Wie bekommt man eine Note für diese Lehrveranstaltung?

13 Tests, je 10 Testpunkte, mind. 5 Testpunkte in mind. 10 Tests, Gründe für nicht-Teilnahme an einem Test (keine Lust, Bus verpasst, Krankheit, ...) machen keinen Unterschied.

Übungstermin: 2a. **Mo** 4. Block,

Falls nötig Verlosung von 20 Plätzen für die Übungsgruppe.

Hinweis auf parallele LV für Zug 1, bei Herrn Goethe: Di 3. Block SU, 4. bzw. 5. Block: Ü

Grundlage der LV: Das Buch "**Java ist eine Sprache**" (in der BHT-Bibliothek oder
im Buchhandel, 40 Euro). Bringen Sie es immer mit. Bin dankbar für Hinweise auf Fehler
Liste bekannter Fehler auf der Netzseite des Buches:
http://public.beuth-hochschule.de/~grude/JavaIstEineSprache/welcome.html.

Fragen an die TeilnehmerInnen:

1. Sie hatten 2 Semester C-Programmierung. Betriebssystem? Compiler?
2. Wer hat schon mal ein Java-Programm geschrieben?
3. Wer hat zu Hause einen Rechner? Alle?
4. Wer hat sich das Buch "Java ist eine Sprache" schon besorgt?

Klausurtermin: Vermutlich am Mo 30.01.2012, 12.15 Uhr, B325 (wird noch bestätigt oder geändert)

Fragen zu Organisation dieser Lehrveranstaltung?

Wie fanden Sie es, in C zu programmieren?
Was war ***leicht***? Was war ***schwer***?
Ist die Sprache C ***standardisiert***?
Wann und wozu benutzt man in einem C-Programm eine ***Funktionsdeklaration*** wie z.B.:
`int add(int, int);` ?
Wie viele Werte gehören (in C) zum Typ `int`?

Wenn jemand Lust hat, 4 verschiedene C-Compiler (gcc, lcc, bcc32, tcc, unter Windows) auszuprobieren oder den C-Standard (ca. 550 Seiten) auswendig zu lernen, siehe:

http://public.beuth-hochschule.de/~grude/TB1-IN1-SWD.zip

3. Jetzt geht es los mit dem Stoff (Abschnittsnummern aus dem Buch)

Es geht in dieser Lehrveranstaltung um 2 Sprachen: **Deutsch** und **Java** (in dieser Reihenfolge).

Rein äußerlich sehen viele Befehle in einem Java-Programm ganz ähnlich aus wie C-Befehle. Aber die "Ideen hinter diesen Befehlen", die Grundkonzepte auf denen sie beruhen, sind bei Java ziemlich anders als bei C.

Ganz allgemein gilt: C ist eine ***maschinennahe*** Sprache. Was ein C-Programm bewirkt hängt (nicht nur vom Programm sondern meistens) auch davon ab, auf was für einer Plattform (Hardware, Betriebssystem) man es ausführen lässt. Der C-Standard legt die Bedeutung von C-Programmen nicht in allen Einzelheiten fest, sondern überlässt vieles der jeweiligen Implementierung, z.B.

- Wie viele Ganzzahltypen gibt es? Nur einen? Oder 2? Oder 8? Oder ...
- Wie viele Werte gehören zum Ganzzahltyp `int`?
- Wie groß ist der größte Wert des Bruchzahltyps `double`?
- Was passiert, wenn man die `double`-Zahl `17.0` durch die `double`-Zahl `0.0` dividiert?

etc.

Dagegen ist Java eine ***maschinenunabhängige*** Sprache und die Sprachbeschreibung legt die Bedeutung von Java-Programmen ***bis in alle Einzelheiten*** ("Bit-genau") fest, z.B.

- Es gibt 5 Ganzzahltypen (namens `byte`, `char`, `short`, `int`, `long`)
- Zum Ganzzahltyp `int` gehören genau 2^{32} (etwa 4.3 Milliarden) Werte
- Der größte Wert des Bruchzahltyps `double` ist etwa gleich `1.797693E308`, seine genaue Darstellung als Dezimalzahl erfordert 309 Dezimalziffern (alle vor dem Dezimal-Punkt, alle genau festgelegt)
- Wenn man die `double`-Zahl `17.0` durch die `double`-Zahl `0.0` dividiert ist das Ergebnis ein spezieller `double`-Wert namens `POSITIV_INFINITY`.

Noch ein paar wichtiger Unterschiede:

Ein C-Programmierer	Ein Java-Programmierer
interessiert sich dafür und weiß, wie seine Programme im Speicher eines bestimmten Rechners aussehen.	sollte abstrakter denken und sich weniger um konkrete Maschinen kümmern.
ist stolz darauf, die ASCII-Codes aller gängigen Zeichen auswendig zu können.	ist stolz darauf, dass man seine Programme lesen kann ohne irgendwelche ASCII-Codes auswendig zu kennen.
kennt ein bestimmtes Compilationsmodell (insbesondere die Arbeitsteilung zwischen Compiler und Binder).	interessiert sich meist nicht dafür, wie oft seine Quellprogramme compiliert werden (gar nicht, einmal, zweimal, ...) und ob es einen Binder gibt.

Jetzt wollen wir noch mal ganz von vorn anfangen und uns (kurz) damit beschäftigen, was ***Programmieren*** eigentlich ist und was die wichtigsten Grundbegriffe der Programmierung sind.

1.1 Programmieren als ein Rollenspiel

Rolle	Tätigkeiten (die meisten beziehen sich auf Programme)
Programmierer	schreibt, übergibt (an den Ausführer)
Ausführer	prüft, lehnt ab/akzeptiert, führt aus
Benutzer	läßt ausführen, ist für Ein-/Ausgabedaten zuständig
Warter	wartet (Programme, nicht auf den Bus)
(Wieder-) Verwender	will wiederverwenden

Sinn der Rollen ***Warter*** und (***Wieder-) Verwender***. Gemeinsam bezeichnen wir den ***Warter*** und den (Wieder-) ***Verwender*** auch als "die ***Kollegen*** des Programmierers".

Besetzung der Rollen, verschiedene Möglichkeiten.

Def.: **Programm**: (S. 3)

Bevor wir uns mit "kleinen Einzelheiten" befassen, ein Versuch, die "großen Ideen" der Programmierung kurz zu beschreiben:

1.4 Die vier wichtigsten Grundkonzepte der Programmierung

Variable (Wertebehälter, Inhalt beliebig oft veränderbar)
Typ (Bauplan für Variablen)

... wird nächstes Mal fortgesetzt

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].

Wichtiger Punkt für die letzten 10 Minuten (anderem Stoff vorziehen):

Zahlensysteme (Dezimalsystem, Binärsystem, Hexadezimalsystem, ...). Wie viele gibt es?

Einfachere Namen: 2-er-System, 3-er-System, ..., 8-er-System, ..., 10-er-System, ..., 16-er-System, ...

All diese Zahlensysteme funktionieren nach demselben System, sie unterscheiden sich nur durch eine einzige Zahl voneinander: Die Basis.

Wenn man die Basis-Zahl b eines Zahlensystems kennt, kann man alles andere daraus ableiten.

Wie viele Ziffern gibt es im 137-er-System?

Die Begriffe Position (oder: Stelle) und Stellenwert (b ist die Basis des Systems):

Positionen	...	3	2	1	0	-1	-2	-3	...
Stellenwerte	...	b^3	b^2	b^1	b^0	b^{-1}	b^{-2}	b^{-3}	...
Zifferern einer Zahl	...								...

1. Übung: Mo 26.09.2012, 4. Block, Raum DE 16a

1. Falls mehr als 22 StudentInnen an dieser Übungsgruppe teilnehmen möchten, 22 Plätze verlosen.

Anmerkung: Die Kapazität der Übungsgruppen (ca. 22 TeilnehmerInnen) wird vor allem durch die ***Kapazität des Betreuers*** begrenzt, ***nicht*** durch die Anzahl der Rechner im SWE-Labor. Diese ***menschliche Betreuungskapazität*** wird ***nicht*** größer, wenn einige StudentInnen auf eigenen Laptops arbeiten, statt die Labor-Rechner zu benutzen (eigene Laptops benötigen eher ein bisschen mehr menschliche Betreuung als die fertig eingerichteten Laborrechner).

2. Einloggen (Noch fehlende Benutzer-Konten einrichten).

3. Meine Netzseite: http://public.beuth-hochschule.de/~grude/

4. Dort die Datei für TB3-IN3 `TipsZumTextPad.pdf` öffnen und den folgenden Tip ausführen: **6. Ein kleines Programm zusammenklicken statt es einzutippen.**

In das erstellte Programm einen Fehler einbauen (z.B. eine Klammer löschen).
Erneut compilieren.

Auf die 1. Zeile der Fehlermeldung doppelklicken. Was passiert dadurch?

Den eingebauten Fehler korrigieren (z.B. die Klammer wieder einfügen).
Erneut compilieren und ausführen lassen.

5. Netzseite zum Buch "Java ist eine Sprache":
http://public.beuth-hochschule.de/~grude/JavaIstEineSprache/

Von dort das Archiv `BspJaSp.zip` herunterladen
und in ein Verzeichnis namens `Z:\BspJaSp` entpacken.

6. Ein Verzeichnis namens `Z:\Java\Hallo` erstellen
und aus `Z:\BspJaSp` alle `Hallo*.java`-Dateien nach `Z:\Java\Hallo` kopieren.

Die drei Java-Quelldateien `Hallo01.java`, `Hallo01A.java` und `Hallo01B.java` mit dem TextPad öffnen, compilieren und ausführen lassen.

Was ist bei diesen drei Quelldateien gleich? Wodurch unterscheiden sie sich?

7. Auf meiner Netzseite die Datei für TB3-IN3 `Aufgaben.pdf` öffnen

Die Seiten 6 und 7 ansehen und den darin beschriebenen Stoff genau verstehen
(ist hoffentlich schon bekannt, aber falls nicht, dann jetzt noch mal lernen).

Die ***Aufgabe 1*** (auf den Seiten 4 und 5) ausdrucken und lösen (in Gruppen zu je 2 Personen).

2. SU Mo 10.10.11

Heute in der Übung: Test 1

1.4 Die vier wichtigsten Grundkonzepte der Programmierung (Fortsetzung)

Variable (Wertebehälter, Inhalt beliebig oft veränderbar) hatten wir schon
Typ (Bauplan für Variablen) hatten wir schon

Was ist eine Variable (in den meisten Programmiersprachen)?

Unterschied einer solchen Variablen mit Variablen in der Mathematik?

Was ist ein Typ?

Unterprogramm (Befehlsfolge mit Namen, der Name ist "ein neuer Befehl". Evtl. Parameter)

Eng verwandte Begriffe: Funktion, Prozedur, Methode

Def.: Eine ***Methode*** ist ein ***Unterprogramm***, das innerhalb einer Klasse vereinbart wurde.

Def.: Ein ***Attribut*** ist eine ***Variable***, die innerhalb einer Klasse vereinbart wurde.

In Java sind ***alle*** Unterprogramme ***Methoden*** (d.h. in Klassen vereinbart).
In C++ gibt es ***Methoden*** und andere ***Unterprogramme*** (die ausserhalb von Klassen vereinbart wurden)

Modul (Behälter für Variablen, Unterprogramme, Typen, ... etc. der aus mind. ***2 Teilen*** besteht, einem ***öffentlichen*** Teil und einem ***privaten*** Teil)
Vorteil von Modulen?
1. Änderungen im privaten Bereich eines Moduls (z.B. ein Unterprogramm entfernen, oder seinen Namen oder seine Parameter ändern, etc.) können den Rest des Programms nicht "durcheinander bringen"

2. Beim Testen muss man bestimmte Fehler nur in einem Modul suchen, statt in allen Modulen.

Nebenbei: Gibt es in C Module?

Zum Abschluss: Das 5. der 4 wichtigsten Grundkonzepte der Programmierung:

Klasse Eine Klasse ist ein Modul und ...

Die Definition einer Klasse wird später vervollständigt und genauer behandelt

1.5 Drei Arten von Befehlen (und wie man sie ins Deutsche übersetzen kann)

Vereinbarung	declaration	"Erzeuge ..." (eine Variable, ein Unterprogramm, einen Typ, ...)
Ausdruck	expression	"Berechne den Wert des Ausdrucks ..."
Anweisung	statement	"Tue die Werte ... in die Wertebehälter ..."

Def.: Ein ***Wertebehälter*** ist entweder eine ***Variable*** oder ein ***Ein-/Ausgabegerät***, z.B. ein Bildschirm, ein Drucker, eine Tastatur, eine Festplatte, ...

Regel: Ein normaler Ausdruck bewirkt nur, dass ***ein Wert berechnet wird***. Er bewirkt nicht, dass der Inhalte irgendeines Wertebehälters verändert wird. Es gibt aber Ausnahmen zu dieser Regel (Ausdrücke mit Seiteneffekt)

Wenn man von einem Befehl nur weiß, zu welcher Art er gehört, weiß man schon sehr viel über ihn.

Beispiele für Java-Befehle und ihre Übersetzug ins Deutsche:

Befehl	**Art**	**Übersetzung ins Deutsche**
int otto;	Ver.	Erzeuge eine Variable namens otto vom Typ int ("baue eine Variable otto nach dem Bauplan int")
int anna = 17;	Ver.	Erzeuge eine Variable namens anna vom Typ int mit dem Anfangswert 17
anna + 3	Aus.	Berechne den Wert des Ausdrucks anna + 3
5 * (otto+anna) - 17	Aus.	Berechne den Wert des Ausdrucks 5 * (otto+anna) - 17
otto = anna + 1;	Anw.	Berechne den Wert des Ausdrucks anna + 1 und tue ihn in die Variable otto.
System.out.print("Hallo");	Anw.	Berechne den Wert des Ausdrucks "Hallo" und tue ihn in die Standardausgabe (das ist meistens der Bildschirm).

Zur Erinnerung: Wer gibt diese Befehle? Wem werden sie gegeben?

(Der ***Programmierer*** gibt sie dem ***Ausführer***).

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 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", von einer Seite einer Gleichung auf die andere). Daraus entstand unser Wort **Algebra**.

5.3 Mit primitiven Werten rechnen (S.103)

Verschiedene Mengen von Zahlen wie ***Mathematiker*** und ***Ingenieure*** sie sehen: S.103, Bild 5.1
Verschiedene Mengen von Zahlen wie ein ***Java-Ausführer*** sie sieht: S.104, Bild 5.2

Wie viele Elemente enthalten die Zahlen-Mengen N, Z, Q, R und C?
(***unendlich*** viele)
Was haben die Mengen N, Z, Q, R und C miteinander zu tun?
(Sie sind ***Teilmengen*** voneinander: $N \subset Z$, $Z \subset Q$, $Q \subset R$, $R \subset C$)

Wie viele Elemente enthalten die Zahlen-Mengen
`byte`, `char`, `short`, `int`, `long`, `float`, `double`?
(***endlich*** viele, die genauen Anzahlen stehen auf S. 93 des Buches)

Was haben die Mengen `byte`, `char`, `short`, `int`, `long`, `float`, `double` miteinandner zu tun?
(Wenig, sie sind ***disjunkt***, d.h. sie haben ***keine*** gemeinsamen Elemente).

In der Mathematik gibt es nur ***eine*** Zahl siebzehn. Diese Zahl ist gleichzeitig
eine natürliche, eine ganze, eine rationale, eine reelle und eine komplexe Zahl.

Ein Java-Ausführer unterscheidet ***sieben*** Zahlen namens siebzehn:
Eine siebzehn vom Typ `byte`, eine siebzehn vom Typ `char`, ..., eine siebzehn vom Typ `double`.
Kurz-Notation: byte-17, char-17, short-17, int-17, long-17, float-17, double-17.

Diese sieben Zahlen ***unterscheiden sich erheblich*** voneinander. Beispiel für Unterschiede:

Beispiel-1: Die Zahl `double-17` darf man einer Variablen des Typs `double` zuweisen,
aber nicht einer Variablen des Typs `int` oder `byte` etc.

Beispiel-2: Wenn man die Zahl double-17 durch die Zahl double-2 dividiert, kommt als Ergebnis
die Zahl double-8.5 heraus.

Wenn man die Zahl int-17 durch die Zahl int-2 dividiert, kommt als Ergebnis
die Zahl int-8 heraus.

Beispiel-3: Wenn man die Zahl float-17 durch die Zahl float-0 dividiert, kommt als Ergenis
eine float-Zahl namens Float.POSITIVE_INFINITY heraus.
Wenn man die Zahl long-17 durch die Zahl long-0 dividiert, kommt kein Ergebnis heraus,
vielmehr wird "eine Ausnahme geworfen". Praktisch bedeutet das häufig (aber nicht immer),
dass die Ausführung des betreffende Programm mit einer Fehlermeldung abgebrochen wird).
Mit Ausnahmen werden uns später genauer befassen.

Allgemein gilt:

Wenn ein Java-Ausführer mit ***Ganzzahlen*** rechnet (d.h. mit Zahlen der
5 Typen `byte`, `char`, `short`, `int`, `long`), dann gelten ganz andere Regeln, als wenn er
mit ***Gleitpunktzahlen*** rechnet (d.h. mit Zahlen der 2 Typen `float`, `double`).

Grundsätzlich gilt:

Wenn der Ausführer mit Ganzzahlen rechnet, ist das Ergebnis seiner Rechnung immer eine Ganzzahl, keine Bruchzahl. Z.B. ist int-7 / int-2 gleich int-3 (und nicht etwa 3.5).

Nur wenn der Ausführer mit Bruchzahlen rechnet, kann das Ergebnis eine Bruckzahl sein. Z.B. ist double-7 / double-2 gleich double-3.5 .

Was passiert, wenn ein Rechenergebnis "zu groß" ist?

S. 112, Bild 5.3 Die Struktur eines Java-Ganzzahltyps

Zum Typ `double` gehören zwei unendliche Werte: `Double.POSITIVE_INFINITY` und `Double.NEGATIVE_INFINITY` (Für den Typ `float` ganz entsprechend). Diese Werte repräsentieren alle "zu großen Zahlen". Sie müssen mit diesen Werten rechnen lerenen (im Kopf, weil Ihr Taschenrechner diese Werte nicht kennt und nicht mit ihnen rechnen kann).

In einem Java-Programm bezeichnen
Literale wie `0` oder `123` oder `587389` immer `int`-Werte,
Literale wie `0.0` oder `12.345` oder `8799.34567` bezeichnen dagegen `double`-Werte.

Es folgen ein paar Beispiele für Rechnungen mit `int` Werten:

```
7 / 2 ist gleich 3
5 / 0 loest eine Ausnahme des Typs ArithmeticException aus (Programmabbruch)
0 / 0 loest eine Ausnahme des Typs ArithmeticException aus (Programmabbruch)
```

Beispiele für Rechnungen mit `double`-Werten:

```
7.0 /  2.0          ist gleich  3.5
5.0 /  0.0          ist gleich  Double.POSITIVE_INFINITY
5.0 / -0.0          ist gleich  Double.NEGATIVE_INFINITY
0.0 /  0.0          ist gleich  Double.NaN
7.0 / (5.0 /  0.0)  ist gleich  0.0
7.0 / (5.0 / -0.0)  ist gleich -0.0
7.0 + (0.0 /  0.0)  ist gleich  Double.NaN
```

Anhand der Aufgaben 2 und 3 sollen sie sich mit den Rechenregeln für `int`-Werte bzw. `float`-Werte vertraut machen. Auf **S. 107** des Buches finden sie zwei Tabellen, die die Multiplikation `*` und die Division `/` für `float`- und `double`-Werte beschreiben.

3. SU Mo 17.10.11

Heute in der Übung: Test 2

Rechnen mit Ganzzahlen und mit Bruchzahlen, kleine Ergänzung

Betrachten Sie folgende Java-Befehle:

```
1    int otto = 1;
2    while (true) {
3       otto = 2 * otto;
4    }
5
6    double emil = 1.0;
7    while (true) {
8       emil = 2.0 * emil;
9    }
```

Wie verändert sich der Wert der Variablen otto? Und wie verändert sich emil?

S. 107, Tabellen für die Multiplikation und Divison von Gleitpunktzahlen, kurz besprechen.

Drei Arten von Befehlen, kleine Ergänzung

Tip: Um einen Befehl richtig einzuordnen sollten Sie sich fragen:

Was befiehlt dieser Befehl dem Ausführer? Etwas zu erzeugen? Oder einen Wert zu berechnen?
Oder die Inhalte bestimmter Wertebehälter (Variablen, Bildschirm etc.) zu verändern?

Typen und Variablen

Def.: Ein ***Typ*** ist ein Bauplan für Variablen

Eine Variablen-Vereinbarung beginnt deshalb immer mit einem Typ.

```
10 // Variablen-Vereinbarungen (mit Initialisierung)
11 int    anna = 17;
12 String bert = "Hallo!";
13
14 // Zuweisungen (keine Vereinbarungen!)
15 anna = 18;
16 bert = "Hello?";
```

2.3. Mehrere Klassen, ein Programm (S. 31)

In Zeile 6 wird eine Methode aufgerufen. Welche?

Was passiert in Zeile 7? In Zeile 8?

Die Punktnotation: **Modul Punkt Element**

`Hallo04.pln(...)`	Die Methode `pln` im Modul `Hallo04`
`Math.sin(...)`	Die Methode `sin` im Modul `Math`
`Integer.MAX_VALUE`	Das Attribut `MAX_VALUE` im Modul `Integer`
`Math.PI`	Das Attribut `PI` im Modul `Math`

Mit dieser Notation ist es also ganz leicht, innerhalb eines Moduls `M1` auf Methoden und Atrribute in einem anderen Modul `M2` zuzugreifen. Die Module `M1` und `M2` können z.B. zwei Klassen sein (später werden wir noch andere Module kennen lernen, die keine Klassen sind).

4 Anweisungen, einfache und zusammengesetzte (S. 46)

(***simple*** and ***compound*** expressions).

Def.: Eine Anweisung ist ***zusammengesetzt***, wenn sie andere ***Anweisungen*** enthalten kann.

Einfache Anweisungen: Zuweisung, Prozeduraufruf (in C-Slang: Aufruf einer `void`-Funktion), `return`, `break`, `continue`, `throw`, `assert`, die leere Anweisung.

Woraus besteht eine Zuweisung? (Aus einem Variablennamen und einem Ausdruck).
Eine `Zweisung` kann aber keine `anderen Anweisungen` enthalten.

Woraus besteht eine `return`-Anweisung? (Nur aus dem Schlüsselwort `return` oder aus `return` und einem Ausdruck).
Eine `return`-Anweisung kann aber keine `anderen Anweisungen` enthalten).

Wieso ist return eine Anweisung? Welchen Wertebehälter verändert sie?

Zusammengesetzte Anweisungen: Die Blockanweisung `{... }`, `if`, `switch`, `while`, `for`, ...

Eine `if`-Anweisung enthält Anweisungen als ihre Teile (sie besteht unter anderem aus Anweisungen).
Wie viele Anweisungen kann eine `if`-Anweisung enthalten? Mindestens? Höchstens?
(Mindestens ***eine***, höchstens ***zwei***).

Wie viele Anweisungen enthält eine `while`-Schleife? (Genau ***eine***).

Zur Entspannung: **Eigenschaften von Qbits (Quanten-Bits)**

1. Mit n "normalen Bits" kann man ***eine*** Zahl (zwischen 0 und 2^n-1) darstellen. Mit n Qubits kann man gleichzeitig bis zu 2^n Zahlen (zwischen 0 und 2^n-1) darstellen und mit ***einer*** Operation kann man alle diese Zahlen "gleichzeitig bearbeiten".

2. Wenn man ein Qubit "ansieht und ausliest", bekommt man nur ***einen*** seiner Werte. Alle anderen Werte gehen dabei unvermeidbar und unwiderruflich verloren.

3. Es ist nicht möglich, ein Qubit (mit all seinen Werten) zu kopieren. Man kann höchstens ***einen*** seiner Werte kopieren.

4. Auf Qubits kann man nur ***umkehrbare Verknüpfungen*** anwenden.

Zur Zeit (2008) erforschen mehrere Tausend Physiker, Informatiker und Ingenieure in mehr als 100 Forschungsgruppen etwa ein Dutzend Möglichkeiten, Qbits zu realisieren (durch ion traps, quantum dots, linear optics, ...).

Eine interessante Einführung in die Quantenmechanik von einer berliner Schülerin:

Silvia Arroyo Camejo: "Skurrile Quantenwelt", Springer 2006, Fischer 2007

Ausdrücke, einfache und zusammengesetzte

Einfache: Literale und Variablen-Namen. Zusammengesetzte Ausdrücke bestehen aus Ausdrücken, Operatoren wie `+`, `-`, `*`, `/`, `%`, `<`, `<=`, `&&`, `||`, `!` ... etc. und runden Klammern.

Beispiele: Aus welchen Teilen bestehen die Ausdrücke in der nachfolgenden Tabelle?

```
// Als "Rohmaterial" vereinbaren wir ein paar Variablen:
int     otto, emil, anna, bert, carl, dora;
boolean fanny, gerd, heinz;
```

<table>
<tr><th>Ausdruck</th><th>Teil 1</th><th>Teil 2</th><th>Teil 3</th></tr>
<tr><td>otto + emil</td><td>otto</td><td>+</td><td>emil</td></tr>
<tr><td>anna + bert + carl + dora</td><td>anna + bert + carl</td><td>+</td><td>dora</td></tr>
<tr><td>anna + bert * carl</td><td>anna</td><td>+</td><td>bert * carl</td></tr>
<tr><td>anna * bert + carl</td><td>anna * bert</td><td>+</td><td>carl</td></tr>
<tr><td>-otto</td><td>-</td><td>otto</td><td></td></tr>
<tr><td>fanny || gerd && heinz</td><td>fanny</td><td>||</td><td>gerd && heinz</td></tr>
<tr><td>fanny && gerd || heinz</td><td>fanny && gerd</td><td>||</td><td>heinz</td></tr>
<tr><td>!heinz</td><td>!</td><td>heinz</td><td></td></tr>
</table>

4. SU Mo 24.10.11

Heute in der Übung: Test 3

Zusammengesetzte Ausdrücke: Fortsetzung

Die Tabelle besprechen, die im 3. SU verteilt und teilweise ausgefüllt wurde:

Ausdruck	Teil 1	Teil 2	Teil 3
`otto + emil`	`otto`	`+`	`emil`
`anna + bert + carl + dora`	`anna + bert + carl`	`+`	`dora`
`anna + bert * carl`	`anna`	`+`	`bert * carl`
`anna * bert + carl`	`anna * bert`	`+`	`carl`
`-otto`	`-`	`otto`	
`fanny || gerd && heinz`	`fanny`	`||`	`gerd && heinz`
`fanny && gerd || heinz`	`fanny && gerd`	`||`	`heinz`
`!heinz`	`!`	`heinz`	

Noch ein Problem mit zusammengesetzten Ausdrücken:

Seien b1 bis b4 boolean-Variablen. Wie ist dann der folgende Ausdruck zu lesen? Welche "impliziten Klammern" muss man sich dazudenken? Welche Regeln gelten dafür?

```
    !  b1   ||     b2   ==   b3     &&    b4
( ( !  b1 ) || ( ( b2   ==   b3  )  &&    b4 ) )
4 1        1    3 2               2          3 4
```

Im Buch, S. 144, findet man eine Tabelle mit allen Operatoren und ihre ***Bindungsstärken***. Man klammert zuerst die Operatoren mit den höchsten Bindungsstärken, dann abwärts weiter:

```
!  ==  &&  ||
```

5 Typen

In Java unterscheidet man 2 Arten von Typen:

Primitive Typen (genau 8 Stück:
`byte, char, short, int, long, float, double, boolean`)
Referenztypen (mehr als 4000 Stück:
`String, StringBuilder, Integer, int[], String[][], Collection<K>`, ...)

5.7 Variablen als Bojen darstellen

Variablen sind das wichtigste Grundkonzept in den meisten Programmiersprachen (es gibt aber auch Programmiersprachen ***ohne Variablen***). Darum brauchen wir ein "genaueres Modell" von Variablen, eine Darstellung von Variablen, die die Anzahl und Art ihrer Bestandteile ganz deutlich macht.

Das **Bild 5.6 auf S. 119** an der Tafel entwickeln, dann kurz im Buch ansehen.

```
StringBuilder otto = new StringBuilder("Hallo!");
|otto|--<78>--[<22>]--["Hallo!"]
```

Def.: Eine Variable besteht aus mindestens ***zwei Teilen*** (**Referenz und Wert**). Einige Variablen haben ein oder zwei zusätzliche Teile (**Name** und/oder **Zielwert**).

Variablen eines primitiven Typs (**primitive Variablen**) bestehen aus
höchstens ***3 Teilen*** (Name, Referenz, Wert).

Variablen eines Referenztyps (**Referenzvariablen**) bestehen aus
höchstens ***4 Teilen*** (Name, Referenz, Wert, Zielwert)

S. 121, Beispiel-02: Der Unterschied zwischen
primitiven Variablen (eines primitiven Typs) und
Referenzvariablen (eines Referenztyps)

S. 122, Beispiel-03: Eine Zuweisung ändert immer den *Wert* einer Variablen!
(und nicht etwa die Referenz oder den Zielwert einer Variablen)

S. 125, Beispiel-07: Die Vergleichsoperatoren `==` und `!=` vergleichen immer *Werte*!
(und nicht etwa Referenzen oder Zielwerte von Variablen)

Zielwerte kann man nur vergleichen, wenn ihr Typ das zuläßt. Und wenn er es zuläßt, muss man zum Vergleichen ***Methoden*** namens `equals`, `compareTo` bzw. `compare` verwenden
(und nicht etwa ***Operationen*** namens `<`, `<=`, `==`, `!=`, `>=`, `>`).

Zur Entspannung: **Englische Vokabeln**: ambiguous, geek, drag etc.	
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***.

Hüllklassen (S. 91)

Zu jedem primitiven Typ (z.B. `int`) gibt es eine entsprechende Hüllklasse (z.B. `Integer`). Die Werte des primitiven Typs kann man in Objekte der Hüllklasse "einhüllen" (später mehr dazu).

5. SU Mo 31.10.11

Heute in der Übung: Test 4

Wiederholung

Stellen Sie die folgenden Variablen als Bojen dar:

```
double dora1  = 1.5;
double dora2  = 2.5;
String sonja1 = new String("ABC");
String sonja2 = new String("123");
```

Welche Teile der Variablen werden durch `dora1 == dora2` verglichen?
Welche Teile der Variablen werden durch `sonja1 != sonja2` verglichen?

Wie kann man die ***Zielwerte*** von `sonja1` und `sonja2` vergleichen?

```
sonja1.equals(sonja2)
```

Wichtige **Regeln**:

In Java sind die ***Zielwerte*** von Variablen immer ***Objekte.***
Ein Objekt ist ein ***Modul.***
In Java enthält jedes Objekt eine Methode namens `equals` (mit ***einem*** Parameter).
Was die `equals`-Methode macht, hängt vom Typ des Objekts ab.

Beispiele:
Die `equals`-Methode von `String`-Objekten vergleicht ***Zielwerte.***
Die `equals`-Methoden von `StringBuilder`-Objekten vergleicht ***Werte.***

Die Übung "Einfach boolean" austeilen und bearbeiten

Sinn der **Aufgabe 4** (Dreiseits) und dieser **Übung**: Häufig ist es viel einfacher, mit `boolean`-Werten zu rechnen (ähnlich wie mit `int`-Werten), statt umständliche `if`-Befehle zu programmieren.

Zur Entspannung: **Nicht-transitive Würfel**

Viele "Vergleichs-Relationen" wie ***istSchneller, istHöher, istGrößer*** etc. sind ***transitiv***, d.h.
wenn A schneller ist als B und B schneller ist als C dann gilt auch: A ist schneller als C.

Betrachten wir noch eine andere ***istBesser***-Relation: Zwei Personen würfeln mehrmals mit zwei Würfeln W1 und W2 gegeneinander. Wer die höhere Augenzahl würfelt, hat den Wurf gewonnen und bekommt einen Punkt. Die Würfel W1 und W2 können unterschiedliche Zahlen auf ihren sechs Seiten haben und somit besser oder schlechter sein. Z.B. ist ein Würfel, der auf allen sechs Seiten eine 6 hat offensichtlich besser als einer, der auf allen sechs Seiten eine 1 hat. Aber was ist mit den folgenden 4 Würfeln?

W1: 0 0 4 4 4 4
W2: 3 3 3 3 3 3
W3: 2 2 2 2 6 6
W4: 1 1 1 5 5 5

In einem Kampf W1 gegen W2 wird W1 im Durchschnitt
2/3 aller Würfe gewinnen (mit 4 zu 3) und
1/3 aller Würfe verlieren (mit 0 zu 3).
W1 ist also (ca. 33 %) besser als W2.

Aus ganz ähnlichen Gründen gilt auch:

W2 ist besser als W3.
W3 ist besser als W4.
Und erstaunlicherweise auch: W4 ist besser als W1.

Quelle: Martin Gardner, "The Colossal Book of Mathematics",
W. W. Norton & Company, ca. 700 Seiten, 35,- US$

8.2 Prozeduren und Funktionen

Auf S. 193 die Tabelle ansehen und besprechen.

3 Programme selbst ausführen (S. 41)

Die Übung **Schleifen (ausführen) 1** austeilen und bearbeiten.

6. SU Mo 07.11.11

Heute kein Test

Prozeduren und Funktionen, Ergänzung

Programmiersprachen, in denen es (beliebig oft veränderbare) Variablen gibt, bezeichnet man auch als **prozedurale Sprachen** (weil es in diesen Sprachen außer Funktionen auch ***Prozeduren*** gibt).

Beispiele: Fortran, Cobol, Algol60, Pascal, C/C++, Java, C#, Perl, Python, ...

Programmiersprachen, in denen es keine (beliebig oft veränderbare) Variablen gibt, bezeichnet man auch als **funktionale Sprachen** (weil es in diesen Sprachen nur ***Funktionen,*** keine Prozeduren gibt).

Beispiele: Lisp, Opal, Scheme, Haskell, Gentle, SQL, XSLT, ...

Beispiele für Programme, die man besonders leicht und elegant in einer ***funktionalen Sprache*** schreiben kann: Compiler.

Für Programme, in denen große Datenstrukturen (z.B. große Reihungen, engl. arrays) häufig verändert werden müssen, sind in der Regel ***prozedurale Sprachen*** besser geeignet.

Fallunterscheidungs-Anweisungen

if-Anweisungen (mit oder ohne `else`)

Aus wie vielen Teilen und was für Teilen besteht eine `if`-Anweisung ***ohne*** `else`?

(sie besteht aus 1 `boolean-Ausdruck` und 1 Anweisung)

Ebenso für eine `if`-Anweisung ***mit*** `else`?

(sie besteht aus 1 `boolean`-Ausdruck und 2 Anweisungen)

Bezeichnungen für die Teile: Bedingung, then-Rumpf, else-Rumpf.

switch-Anweisungen

Sie fahren auf eine kleine Insel und dürfen nur `if-Anweisungen` oder nur `switch`-Anweisungen mitnehmen. Welche würden Sie wählen?

Merke: `switch`-Anweisungen sind "elegante Abkürzungen für bestimmte `if`-Anweisungen", aber notfalls geht es auch ohne `switch`, nur mit `if` .

S. 65, Aufgabe-01: `char`-Variable `c`, Meldung, je nachdem ob `'('`, `')'`, `'['`, `']'` oder anderes Zeichen

S. 65, Beispiel-01: Lösung mit `if`
S. 65, Beispiel-02: Lösung mit `switch`

Was darf in C in den runden Klammern nach dem Schlüsselwort "switch" stehen?

(ein Ausdruck eines beliebigen Ganzzahltyps wie short, int, long, char, ...)

In Java darf dort ein Ausdruck eines "kleinen Ganzzahltyps" stehen (d.h. der Typ `long` ist verboten).

Seit Java 7 darf dort auch ein `String`-Ausdruck stehen.

Was darf nach dem Schlüsselwort "case" stehen?

(Ein ***konstanter Ausdruck***, d.h. ein Ausdruck, dessen Wert man schon ***beim Lesen des Programm-Textes*** berechnen kann, nicht erst dann, wenn das Programm ***ausgeführt*** wird).

S. 67, Beispiel-05: Eine `switch`-Anweisung mit "komplizierteren konstanen Ausdrücken nach `case`"

Reihungen (engl. : arrays)

S. 150, Beispiel-01
S. 150, Beispiel-02
S. 151, Beispiel-03
S. 151, Beispiel-04
S. 152, Beispiel-05

Übung Schleifen (programmieren) 2 austeilen und bearbeiten.

Zur Entspannung: **Niklaus Wirth** (geb. 1934 in Winterthur, Schweiz)

Diplom 1959 an der ETH Zürich, M.Sc. 1960 an der Laval University, Canada und 1963 Ph.D. an der UCL Berkley. Wirth erhielt 1984 den Turing-Award. Wichtige Programmiersprachen:

1955	Fortran, Cobol	Erste höhere Programmiersprachen	
1960	Algol60		
1965	Algol-W	Besseres Algol	(von Wirth)
1968	Algol68	2-Stufen-Grammatiken., Bojen	
1970	C	Maschinennahe höhere Sprache	
1972	Pascal	Strukturierte Programmierung	(von Wirth)
1975	Modula	Module	(von Wirth)
1980	Modula-2	Module, Nebenläufigkeit	(von Wirth)
1980	Ada	Module, Nebenläufigkeit, Schablonen	
1984	C++	C mit Klassen, Schablonen	
1990	Oberon	Klassen	(von Wirth)
1994	Java	Plattformunabhängigkeit	
1995	Ada	Klassen	

Sehr gut: Er hat immer wieder von vorn angefangen.
Schlecht: Er hat immer wieder von vorn angefangen.

7. SU Mo 14.11.11

Heute in der Übung: Test 5

for-i- und for-each-Schleifen

Aus wie vielen Teilen besteht eine `for`-Schleife und wie heissen diese Teile?
S. 75, Bild 4-1

Speziell zum Bearbeiten von ***Reihungen*** (engl.: arrays) und ***Sammlungen*** (engl.: collections) gibt es (seit Java 5) auch **for-each-Schleifen**:

S. 153, Beispiel-06

Regel: Wenn eine `for-each`-Schleife ***möglich*** ist, sollten Sie sie nehmen
(sonst erfolgt möglicherweise ein Punktabzug!).

Referenzvariablen und der Wert null

Der Wert einer Referenzvariablen R kann die Referenz eines Objeks Z sein.
In diesem Fall bezeichnen wir Z auch als den Zielwert von R.

Der Wert einer Referenzvariablen R kann aber auch gleich `null` sein.
In diesem Fall hat die Variable R ***keinen Zielwert***.

Statt `null` fände ich die Bezeichnung `NaR` (wie "not a reference") besser, um die Ähnlichkeit zu den `NaN`-Werten bei Gleitpunkttypen deutlich zu machen.

7.1 Reihungen vereinbaren und als Bojen darstellen

Jede Reihung hat einen ***Typ*** (z.B. ***Reihung-von-long-Variablen***) und
einen ***Komponententyp*** (im Beispiel: ***long***).
Reihungen mit einem ***primitiven Komponententyp*** sehen ein bisschen anders aus als
Reihungen mit einem ***Referenztyp als Komponententyp***.

Merke: Alle Reihungstypen sind ***Referenztypen*** (keine primitiven Typen)

S. 155, Beispiel-01: Vereinbarung einer Reihung mit einem
primitiven Komponententyp (`long`)

Von welchem Typ ist die Variable `lr01`? (Vom Typ ***Reihgung-von-long-Variablen***)

S. 156, Bild 7.1 Die Reihung `lr01` in ausführlicher Bojendarstellung

S. 157, Bild 7.2 Die Reihung `lr01` in vereinfachter Bojendarstellung

In der vereinfachten Darstellung darf man die (unveränderbare) `length`-Variable und die ***Referenzen der Komponenten-Variablen*** (im Beispiel sind das die Referenzen `<35>` bis `<39>`) weglassen (aber nicht die Referenz der Reihungsvariablen, im Beispiel ist das `<45>`).

S. 157, Beispiel-02: Vereinbarung einer Reihung mit einem
Referenztyp als Komponententyp (`StringBuilder`)

Von welchem Typ ist die Reihung `sr01`? (Vom ***Typ Reihung-von-String-Variablen***)

Die Komponenten dieser Reihung sind Referenzvariablen (genauer: `String`-Variablen) ***ohne Namen***.
Anstelle eines Namens hat jede Komponente einen Index.

Merke:
Die ***Referenzen*** und ***Werte*** der Komponenten liegen ***innerhalb*** der Reihung.
Die ***Zielwerte*** der Komponenten liegen ***ausserhalb*** der Reihung!

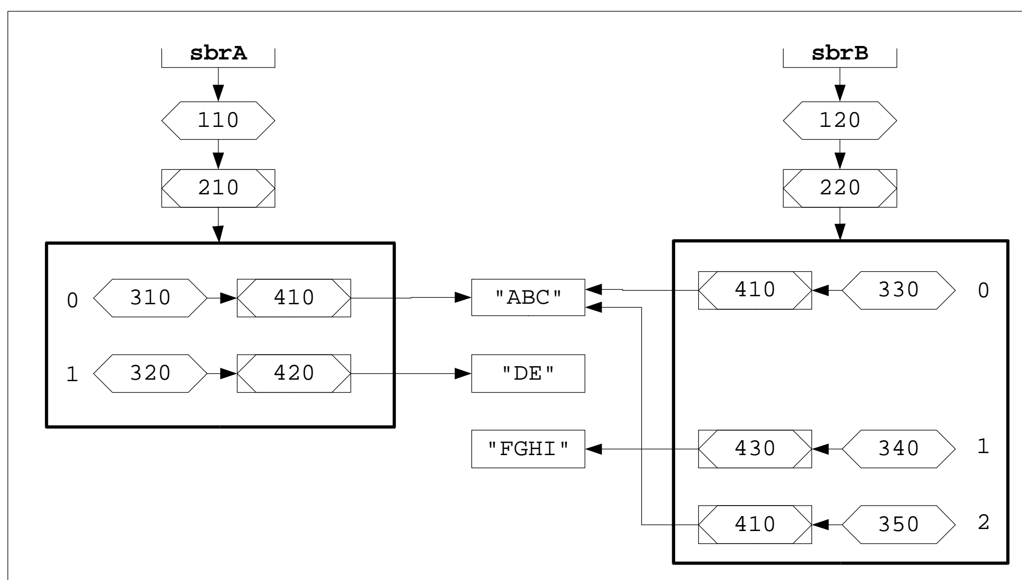
Daraus folgt: Ein Zielwert kann gleichzeitig zu ***mehreren*** Reihungen und
mehrmals zur selben Reihung gehören.

Zur Erinnerung: Eine Referenzvariable (eine Variable eines Referenztyps) kann den Wert `null` haben (dann hat sie ***keinen Zielwert***) oder einen anderen Wert (dann ***hat sie*** einen Zielwert).
In Java sind die ***Zielwerte*** von Referenzvariablen immer ***Objekte***.

Beispiel: Ein `StringBuilder`-Objekt welches gleichzeitig in zwei Reihungen enthalten ist (und in einer Reihung sogar zweimal):

```
 1  StringBuilder[] sbrA = {
 2     new StringBuilder("ABC"),
 3     new StringBuilder("DE"),
 4  };
 5
 6  StringBuilder[] sbrB = {
 7     sbrA[0],
 8     new StringBuilder("FGHI"),
 9     sbrA[0],
10  }
```

Die Reihungen `sbrA` und `sbrB` als Bojen dargestellt:



Hier sieht man 2 Reihungsobjekte (fett umrandet) und
3 `StringBuilder`-Objekte (`["ABC"]`, `["DE"]` und `["FGHI"]`).

1. Wie viele Variablen werden hier insgesamt dargestellt?

2. Wie viele dieser Variablen haben einen (einfachen) Namen?

3. Geben Sie von jeder Variablen an, zu welchem Typ sie gehört ("nach welchem Bauplan sie gebaut wurde").

4. Mehrere Variablen haben gleiche Werte. Wie viele Variablen sind das?

5. Was kann man über die ***Zielwerte*** dieser Variablen (die gleiche Werte haben) sagen?

Anmerkung: Die `StringBuilder`-Objekte sind hier nur angedeutet und nicht realistisch dargestellt.
Jedes `StringBuilder`-Objekt ist ein ***Modul***, der unter anderem folgendes enthält:
60 öffentliche Methoden
4 private Methoden
2 "halb private" Variable, davon eine names `value` vom Typ `char[]` (in dieser Reihung stehen die Zeichen des `StringBuilder`-Objekts, aber man kann von außen nicht direkt darauf zugreifen).

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.

Was bezeichnet der Name einer Variablen?

Aus wie vielen und welchen Teilen besteht eine Variable mindestens? (2, Referenz, Wert)
Aus wie vielen und welchen Teilen besteht eine Variable höchstens? (4, plus Name, Zielwert)

Welcher Teil der Variablen ist gemeint, wen wir ihren Namen in den Befehlen eines Java-Programms erwähnen?

```
11  String st1 = new String("Hallo!);
12  String st2 = null;
13                              // Welchen Teil von st1 bezeichnet der Name "st1"?
14  ... st1.equals("Sonja") ... // Den Zielwert
15  ... st1 == st2 ...          // Den Wert
16  st1 = "Sonja";              // Die Referenz
17  pln(st1);                   // Den Wert
18                              // Den Zielwert (sonst)
```

7.2. Die Erzeugung einer Reihung in 3 Schritten (S. 161)

8. SU Mo 21.11.11

Heute in der Übung: Test 6

Klassen und Objekte (Beginn eines größeren Kapitels)

Zur Erinnerung: Was ist ein Modul? (Ein Behälter für ..., mind. 2 Teile: öffentlich, privat)

Def.: Eine Klasse ist ein ***Modul*** und ein ***Bauplan für Module***.

Beispiel-M für die Benutzung der Klasse `Character` als ***Modul***:

```
1 char meinZeichen = EM.liesChar();
2 ... Character.isDigit(meinZeichen) ...
```

In Zeile 2 wird die Funktion `isDigit` aufgerufen, die sich im Modul `Character` befindet. Sie liefert `true`, wenn ihr Parameter eine Ziffer darstellt. Im Unicode gelten zur Zeit 350 Zeichen als Ziffern (d.h. `isDigit(c)` liefert `true`). Einige davon werden im Folgenden dargestellt (aber für einige enthält der Font ***Arial Unicode MS*** offenbar keine passende Darstellung):

0 1 2 3 4 5 6 7 8 9 ٩ ٨ ٧ ٦ ٥ ٤ ٣ ٢ ١ ٠ ۰ ۱ ۲ ۳ ۴ ۵ ۶ ۷ ۸ ۹

߉ ߈ ߇ ߆ ߅ ߄ ߃ ߂ ߁ ߀ ० १ २ ३ ४ ५ ६ ७ ८ ९ ০ ১ ২ ৩ ৪ ৫ ৬ ৭ ৮ ৯

੦ ੧ ੨ ੩ ੪ ੫ ੬ ੭ ੮ ੯ ૦ ૧ ૨ ૩ ૪ ૫ ૬ ૭ ૮ ૯ ୦ ୧ ୨ ୩ ୪ ୫ ୬ ୭ ୮ ୯

௧ ௨ ௩ ௪ ௫ ௬ ௭ ௮ ௯ ౦ ౧ ౨ ౩ ౪ ౫ ౬ ౭ ౮ ౯ ೦ ೧ ೨ ೩ ೪ ೫ ೬ ೭ ೮ ೯

൦ ൧ ൨ ൩ ൪ ൫ ൬ ൭ ൮ ൯ ๐ ๑ ๒ ๓ ๔ ๕ ๖ ๗ ๘ ๙ ໐ ໑ ໒ ໓ ໔ ໕ ໖ ໗ ໘ ໙

༠ ༡ ༢ ༣ ༤ ༥ ༦ ༧ ༨ ༩

0 1 2 3 4 5 6 7 8 9

Beispiel-B1 für die Benutzung der Klasse `Character` als ***Bauplan***:

```
3 ... new Character(meinZeichen) ...
```

Hier wird dem Ausführer befohlen, einen neuen ***Modul*** nach dem Bauplan `Character` zu bauen. Im Bauplan steht, welche Methoden und Variablen in diesen neuen Modul eingebaut werden müssen (es sind 13 öffentliche Methoden und 0 Variablen).

Module, die man (wie hier gezeigt) mit dem `new`-Befehl erzeugen läßt, werden meist als ***Objekte*** oder als ***Instanzen*** (der Klasse `Character`) bezeichnet.

Beispiel-B2 für die Benutzung der Klasse `Character` als ***Bauplan***:

```
4 Character deinZeichen = null;
```

Hier wird dem Ausführer befohlen, eine ***Variable*** vom Typ `Character` zu bauen. Diese Variable kann im Prinzip auf ein `Character`-Objekt zeigen, wird aber erstmal mit dem Wert `null` initialisiert und zeigt somit erstmal ***nicht*** auf ein Objekt.

Beispiel-B3 für die Benutzung der Klasse `Character` als ***Bauplan***:

Die vorigen beiden Beispiele kann man auch kombinieren, etwa so:

```
5 Character deinZeichen = new Character(meinZeichen);
```

Hier wird dem Ausführer befohlen, einen neuen Modul (oder: ein neues Objekt) nach dem Bauplan `Character` zu bauen und eine Variable namens `deinZeichen` zu erzeugen, die auf das neue Objekt zeigt (oder: die das neue Objekt als ***Zielwert*** hat).

Diese Seite ausdrucken und während der Vorlesung austeilen!

Jetzt das Gleiche nochmal, aber mit mehr Details:

Vergleich: Eine ***Klasse*** (in Java) ist eine Art ***Kombiwerkzeug***, ähnlich wie ein Schraubenzieher, der gleichzeitig ein Kugelschreiber ist. Eine Klasse ist gleichzeitig ein Modul und ein Bauplan für Module.

Def.: Eine **Methode** ist ein ***Unterprogramm***, welches in einer Klasse vereinbart wurde.

Def.: Ein **Attribut** (engl. field) ist eine ***Variable***, die in einer Klasse vereinbart wurde.

9 Klassen (S. 201)

S. 203, Beispiel-01: Die Klasse `Zaehler01`

Wie viele Dinge gehören zum ***Modulaspekt*** der Klasse `Zaehler01` und wie heißen diese Dinge?

Wie viele Dinge gehören zum ***Bauplanaspekt*** der Klasse `Zaehler01` und wie heißen diese Dinge?

Wie erkennt man beim Lesen einer Klasse einer Konstruktor?

Die Vereinbarung sieht aus wie die einer Methode, aber ohne Ergebnis-Typ.
Konstruktoren heißen genau wie die Klasse, zu der sie gehören.

Wenn man mit new ein neues Objekt erzeugen läßt, muss man unmittelbar hinter new immer einen Konstruktor aufrufen, z.B. so:

```
... new Zaehler(123L) ...
```

S. 204, Bild 9.1 Der Modulaspekt der Klasse `Zaehler01`

S. 205, Beispiel-02: Drei `Zaehler01`-***Variablen*** und drei `Zaehler01`-***Objekte*** werden erzeugt

S. 205, Bild 9.2: Die Variablen `zafer` ud `zelia` als Bojen

Den Befehl in Zeile 26 schrittweise ausführen

Was für eine ***Art von Befehl*** ist das? (Eine Vereinbarung, genauer: eine Variablen-Vereinbarung)

Der `new`-Befehl baut ein neues Objekt und initialisiert alle Attribute (Variablen) ***standardmäßig.***

S. 206, Bild 9.3

Dann wir der Konstruktor ausgeführt und kann die Initialwerte der Variablen noch ändern.

S. 206, Bild 9.4

Der `new`-Befehl liefert eine ***Referenz,*** die auf das neu erzeugte Objekt zeigt.

Zur Entspannung: **Selbstzutreffende ("autologische") Worte**

Es gibt Worte, die offenbar auf sich selbst zutreffen, z. B. ***Hauptwort*** (ist ein Hauptwort), ***deutsch*** (ist ein deutsches Wort), ***English*** (ist ein englisches Wort), ***kurz*** (ist ein ziemlich kurzes Wort) und ***dreisilbig*** (ist dreisilbig). Solche Worte bezeichnen wir hier als **selbstzutreffend**.

Worte, die nicht auf sich selbst zutreffen, bezeichnen wir als **selbstunzutreffend**, z. B. ***zweisilbig*** (ist dreisilbig), ***Tätigkeitswort*** (ist ein Hauptwort) und ***englisch*** (ist ein deutsches Wort).

Frage: Ist jedes Wort entweder selbstzutreffend oder selbstunzutreffend? Was ist mit dem Wort selbstunzutreffend? Ist es selbstzutreffend oder selbstunzutreffend?

Wenn noch Zeit ist:

9.2 Eine Klasse mit Attributen eines Referenztyps (S. 211)

S. 211, Beispiel-04: Ein Bauplan für etwas kompliziertere Objekte

Wie viele Konstruktoren werden in der Klasse `Person00` vereinbart?

Elemente sind alle Dinge, die in einer Klasse vereinbart wurden, ***ausser den Konstruktoren***.

Wie viele Elemente werden in der Klasse `Person00` vereinbart und wie heißen sie?

9. SU Mo 28.11.11

Heute in der Übung: Test 7

Klausur: Mi 01.02.2012, 18-20 Uhr, Beuth-Saal

In der 10. Übung am 05.12.2011: Nach dem Test kommt ein Mensch, der sich auf eine Prof-Stelle an der Beuth-Hochschule bewirbt, und hält im Raum D17 eine **Probevorlesung** (Thema: "Generische Datentypen in Java", sehr wichtig für die Klausur!). Nach der Vorlesung werden Sie nach Ihrem Eindruck gefragt (soll der Bewerber eingestellt werden oder nicht?). Damit hat Ihre Meinung einen wichtigen Einfluß auf die Zukunft der Beuth-Hochschule.

10.2 Objekte mit Hilfe von Bojen genauer verstehen (S. 228)

Druckfehler: Im **Beispiel-01**, Zeile 1, sollte es `"Guten Morgen!"` heissen (statt `"Guten Tag!"`)

S. 228, Bild 10.1 Zwei Bojen für dieselbe Variable `s01` besprechen

S. 230, Beispiel-02: `String`-Variablen ohne und mit `new` initialisieren

Haben wir letztes Mal schon kurz besprochen, hier noch mal ein vollständiges Beispiel

10.3 Die Klasse StringBuilder

Ein `StringBuilder`-Objekt enthält eine Zeichenkette und viele Methoden, mit der man die Zeichenkette verändern kann.

Die Zeichenkette steht in einer privaten Variablen vom Typ `char[]`. Die Länge dieser Reihung bezeichnet man als die ***Kapazität*** (engl: capacity) des Objekts. Die Anzahl der (mit char-Werten) belegten Komponenten bezeichnet man als ***Länge*** (engl: length) des Objekts.

Die schnellsten Änderungsmethoden sind die `append`-Methoden. Methoden wie `insert`, `replace` und `delete` müssen evtl. viele Komponenten der `char[]`-Reihung verschieben.

10.4 Die Klasse ArrayList

Die Objekte dieser Klasse haben Ähnlichkeit mit ***Reihungen,*** enthalten aber (anders als Reihungen) viele "komfortable Methoden" zum Einfügen, Suchen und Löschen von Komponenten.

Wichtigster Unterschied zwischen Reihungen und `ArrayList`-Objekten:
"Reihungen sind aus ***Beton***", aber "`ArrayList`-Objekte sind aus ***Gummi***".

Die Klasse `ArrayList` gehört zu den sog. ***Sammlungsklassen*** (engl: collection classes) und die Objekte der Klasse `ArrayList` werden auch als ***Sammlungen*** (engl: collections) bezeichnet.

Aus einer Sammlung einer bestimmten Klasse (z.B. ArrayList) läßt sich ganz leicht eine Sammlung einer anderen Sammlungsklasse (z.B. `TreeSet` oder `HashSet` etc.) mit denselben Komponenten erzeugen.

10.6 Die Klassen BigInteger und BigDecimal

Mit einem Objekt der Klasse `BigInteger` kann man eine praktisch beliebig große Ganzzahl darstellen (Zahlen mit 10 Millionen Ziffern sind kein Problem).

Mit einem Objekt der Klasse `BigDecimal` kann man einen praktisch beliebig großen und genauen Dezimalbruch darstellen (Dezimalbrüche mit 10 Millionen Ziffern vor und 10 Millionen Ziffern nach dem Punkt sind kein Problem).

Kommerzielle Berechnungen (wo es um Geldbeträge geht) sollte man auf keinen Fall mit Gleitpunktzahlen (vom Typ `float` oder `double`) durchführen, sondern mit Objekten der Klassen `BigInteger` bzw. `BigDecimal`.

10.7 Die Klasse Formatter und die printf-Methode

Die Funktion `printf` in C/C++ ist sehr nützlich und leistungsfähig, hat aber auch ein paar Schwächen (z.B. können falsche `printf`-Befehle zu einem Programmabsturz führen und
man kann mit der `printf`-Funktion kein Daten von selbst definierten Typen formatieren, d.h. man kann `printf` nicht "erweitern").

Die Methode `format` (alias `printf`) in Java ist ähnlich nützlich und leistungsfähig wie die C/C++-Funktion, und hat nicht deren Schwächen.

Auf meiner Netzseite gibt es 2 Applets (`eTeachMePrintf` und `PrintfApplet`), mit denen man den Java-Befehl `printf` ausprobieren und erforschen kann.

12 Klassen erweitern (beerben)

Motivation: Wir haben eine "alte und bewährte" Klasse K1 und brauchen eine Variante K2 davon.
Was tun?

K1 ändern?	(Auf keinen Fall)
K1 kopieren und die Kopie ändern?)	(Auf keinen Fall)
K1 zu K2 erweitern?	(Auf jeden Fall!)

S. 278, Beispiel-01: Die ("alte und bewährte") Klasse `Person01`
S. 278, Bild 21.1: Der Modulaspekt der Klasse `Person01`
S. 279, Bild 12.2: Ein `Person01-Objekt`

S. 280, Beispiel.02: Die Klasse `Person02,` eine Erweiterung der Klasse `Person01`
S. 281, Bild 12.3: Der Modulaspekt der Klasse `Person02`
S. 281, Bild 12.4: Ein `Person02-Objekt` (als "Zwiebel" dargestellt)

Zur Entspannung: **Charles Babbage** (1791-1871)

Forschte auf verschiedenen Gebieten. Unternahm mehrere Versuche, mechanische Rechenmaschinen zu bauen. Keine davon wurde funktionstüchtig, aber seine Pläne und Überlegungen dazu waren wichtige Stationen auf dem Weg zu Computern.

1823 Difference Engine no. 1 (in die Entwicklung floßen 17000 Pfund, was etwa dem Preis von zwei Schlachtschiffen entsprach)
1833 Analytical Engine (die sollte sogar schon programmierbar werden)
1847 Difference Engine no. 2 (wurde ab 1985 im Science Museum in London genau nach den Plänen von Babbage gebaut und funktioniert)

Ada Byron, Lady Lovelace (1815-1852), Tochter des berühmten Dichters Lord Byron, arbeitete mit an der Analytical Engine und gilt seitdem als erste Programmiererin. Nach ihr ist die Sprache Ada benannt (ANSI/MIL-STD-1815, nach ihrem Geburtsjahr).

Weitere Erfindungen von Charles Babbage: Kuhfänger (für Lokomotiven, z. B. im Wilden Westen). Er knackte als erster eine Vignére-Verschlüsselung (die vorher als sicher galt). Schrieb das Buch ***Economy of machinery and manufactures***, eine Analyse des Frühkapitalismus und wichtige Quelle für Karl Marx.

10. SU Mo 05.12.11

Heute in der Übung: Test 8

Danach: Probevorlesung eines Bewerbers auf eine Prof-Stelle an der Beuth Hochschule, im Raum D17

12.2 Allgemeine Regeln zum Beerben und Vererben (S. 283)

S. 285, Bild 12.7: Ein Typgraf mit vielen Typen (und noch mehr Auslassungen)

Erbregel-1 bis **-6** besprechen

Aus diesen Regeln folgt: Alle Java-Klassen bilden einen ***Baum*** mit der Klasse `Object` an der Wurzel.

Hinweis: Der Graf im **Bild 12.7** heißt ***Typgraf*** (und nicht ***Klassengraf***) weil es in Java ausser Klassen auch noch andere (Referenz-) Typen gibt, nämlich

Reihungstypen (wie `int[]` oder `String[]` oder `StringBuilder[][][]` etc.)
Generische Typen (wie `ArrayList<Integer>` oder `ArrayList<String>` etc.)
Schnittstellentypen (wie `Serialzable`, `Cloneable`, `Comparable` etc.)

Diese "anderen Typen" sind auch Untertypen von `Object` und können in einen Typgrafen eingezeichnet werden.
Die 8 ***primitiven Typen*** sind nicht Untertypen von `Object` und haben ***keinen Platz*** in einem Typgrafen.

S. 286, Def. *Typgraf*

Die 4 Begriffe ***direkte Unterklasse***, ***Unterklasse***, ***direkte Oberklasse***, ***Oberklasse***

Noch eine Motivation dafür, Unterklassen zu vereinbaren

Angenommen wir haben vor, mehrere Klassen ("... Baupläne für Objekte") zu schreiben, die "sich überlappen", d.h. bestimmte Elemente kommen in jeder von diesen Klassen vor. Statt diese "gemeinsamen Elemente" in jeder Klasse erneut zu vereinbaren, vereinbaren wir eine Oberklasse, die nur die gemeinsamen Elemente enthält, und erweitern diese Oberklasse dann mehrmals.

12. 3 Ein größeres Beispiel für Beerbung (S. 286)

S. 286, Bild 12.8 Ein Typgraf mit 6 Typen

5 dieser Typen sind Klassen, deren Programmierung wir jetzt "nachvollziehen" wollen.

Mit den Objekten der 5 Klassen soll man geometrische Figuren wie Rechtecke, Quadrate, Ellipsen, Kreise etc. darstellen können (z.B. in einem grafischen Editor). Jede dieser Figuren hat einen Mittelpunkt. Also vereinbaren wir zuerst eine Klasse `E01Punkt` (S. 287)

Wie viele Elemente gehören zum ***Modulaspekt*** der Klasse `E01Punkt?` (0)

Wie viele und was für ***Attribute*** werden in jedes `E01Punkt`-Objekt eingebaut? (2, `x`, `y`)

Wie viele und was für ***Methoden*** werden in jedes `E01Punkt`-Objekt eingebaut? (4, `urAbstand`, ...)

Wie viele Konstruktoren werden in der Klasse `E01Punkt` vereinbart? (einer)
Wie viele Parameter hat der Konstruktor? (2) Wie heißen die Parameter? (x, y)
Was ist merkwürdig an diesen Namen? (Sie stimmen mit den Namen der Attribute x, y überein)

Jedes Objekt enthält eine Variable namens `this`, die auf das Objekt selbst zeigt.

Im Kontruktor bezeichnet `this.x` das Attribut x und `x` (ohne `this` davor) bezeichnet den Parameter x.

Was macht der Konstruktor mit seinen Parametern x und y? (Er initialisiert damit die Attribute x und y)

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

Der Werwolf Ein Werwolf eines Nachts entwich / von Weib und Kind und sich begab

S. 288, die Klasse E01Rechteck

Was besagt oder bewirkt die Zeile 32 (`class E01Rechteck extends E01Punkt`)?

Wie viele Attribute werden in der Klasse `E01Rechteck` vereinbart und wie heißen sie?
(2, `seiteX, seiteY`)

Wie viele Attribute werden in jedes `E01Rechteck`-Objekt eingebaut und wie heißen sie?
(4, `x`, `y`, `seiteX, seiteY`)

Ebenso wie für Attribute auch für Methoden.

Aufgabe: Vereinbaren Sie eine Klasse namens `E01Quadrat` als Erweiterung der Klasse `E01Rechteck`.

11. SU Mo 12.12.11

Heute in der Übung: Test 9

Die Probevorlesung von Prof. Funk in der Übung am 05.12.11

Wie fanden Sie diese Vorlesung? Gute Seiten? Weniger gute Seiten?

Hier eine kleines Ergänzung zum Thema Generizität:

1. Zuerst gab es ***ungetypte*** Programmiersprachen (z.B. Assemblersprachen).
Vorteil: Der Programmierer durfte alles, der Compiler/Assembler hat (fast) nichts verboten.
Nachteil: Der Compiler/Assembler hat (fast) keine Flüchtigkeitsfehler des Programmierers entdeckt.

2. Dann hat man ***getypte*** Programmiersprachen erfunden
Vorteil: Der Compiler konnte viele Flüchtigkeitsfehler des Programmierers entdecken.
Nachteil: Bestimmte Konstrukte musste man mehrmals hinschreiben, weil sie "nur für einen bestimmten Typ funktionierten", z.B.
ein Unterprogramm zum Suchen in einer Reihung mit `String`-Komponenten und
ein Unterprogramm zum Suchen in einer Reihung von `int`-Komponenten etc.

3. Da hat man ***generische Konstrukte*** erfunden, in denen anstelle von festen Typnamen wie `String` und `int` etc. auch ***Variablen für Typen*** vorkommen dürfen.
Beispiel für ein solches generisches Konstrukt:
ein Unterprogramm zum Suchen in einer Reihung von T-Komponenten (wobei T eine Typvariable ist).
Dieses Unterprogramm darf man
auf Reihungen von `String`-Komponenten und
auf Reihungen von `int`-Komponenten etc. anwenden.

Generische Konstrukte dienen also dazu, "unangenehme Härten" eines Typsystems "ein bisschen aufzuweichen" und so Vorteile von ungetypten und von getypten Sprachen zu kombinieren.

Ein wichtiger Unterschied zwischen generischen Klassen in Java und in C++?

Wenn man in einem C++-Programm die Typen
`Vector<String>`, `Vector<Punkt>` und `Vector<Person>` verwendet, dann
baut einem der Compiler ***drei Kopien*** der Klasse `Vector<T>` in das Maschinenprogramm ein.

Wenn man in einem Java-Programm die Typen
`ArrayList<String>`, `ArrayList<Punkt>` und `ArrayList<Person>` verwendet, dann
baut einem der Compiler nur ***eine Kopie*** der Klasse `ArrayList<T>` in das Maschinenprogramm ein.

Generische Klassen in C++ sind ***sehr mächtig*** und ***kompliziert*** zu handhaben.
Generische Klassen in Java sind ***weniger mächtig*** und ***viel einfacher*** zu handhaben.

Noch eine Motivation dafür, Unterklassen zu vereinbaren

Sei `OK` eine Oberklasse von `UK` (und somit `UK` eine Unterklasse von `OK`). Dann gilt:

Überall in einem Java-Programm, wo ein `OK`-Objekt verlangt wird (z.B. als Parameter einer Methode) darf man auch ein `UK`-Objekt angeben.

Begründung:
Da die Klasse `UK` alle Elemente der Klasse `OK` geerbt hat,
hat ein `UK`-Objekt alle Elemente eines `OK`-Objekts (und evtl. noch ein paar mehr).
Also "kann ein `UK`-Objekt alles, was ein `OK`-Objekt kann".

Wichtiges Beispiel: Was für Objekte kann man in folgender Reihung aufbewahren? **(S. 295)**

```
E01Punkt[] otto = new E01Punkt[150];
```

`String`-Objekte? (Nein!) `Double`-Objekte? (Nein?) `E01Punkt`-Objekte (Ja, natürlich)
Was noch? (`E01Rechteck`-Objekt, `E01Ellipse`-Objekte, `E01Kreis`-Objekte)

Objekte in Zwiebeldarstellung

Zur Erinnerung: `Object <- E01Punkt <- E01Rechteck <- E01Quadrat`

Ein Objekt einer Klasse wie `E01Quadrat` kann man sich als eine "Zwiebel mit mehreren Schichten" vorstellen **(S. 291, Bild 12.9)**

In jeder "Schicht" können Attribute vereinbart sein. Das ist der Grund für folgende allgemeine

Regel für Konstruktoren: Der erste Befehl in ***jedem*** (!) Konstruktor `K` ruft einen Konstruktor der direkten Oberklasse auf (damit der die Attribute der "tiefer innen liegenden Zwiebelschichten" initialisieren kann). Danach kann `K` dann die Attribute "seiner Schicht" initialisieren.

Konkret: Was macht der erste Befehl in jedem `E01Quadrat`-Konstruktor?
(er ruft einen Konstruktor der Klasse `E01Rechteck` auf)
Was macht der erste Befehl in diesem `E01Rechteck`-Konstruktor?
(er ruft einen Konstruktor der Klasse `E01Punkt` auf)
Was macht der ertste Befehl in diesem `E01Punkt`-Konstruktor?
(er ruft einen Konstruktor der `Object` auf)
Was macht der erste Befehl in diesem `Object`-Konstrukor?
(nichts, weil Object keine Oberklasse und keine Attribute hat)

Wenn man einen Konstruktor programmiert, kann man obige Regel ***selbst "erfüllen"***, indem man als ersten Befehl im Rumpf einen Konstruktor der direkten Oberklasse aufruft, z.B. so:

```
super(17, "ABC", true)
```

Wann wird dieser Befehl vom Ausführer akzeptiert bzw. abgelehnt?
(akzeptiert, wenn es in der direkten Oberklasse einen Konstruktor mit 3 Parametern der Typen `int`, `String` und `boolean` gibt, sonst abgelehnt)

Wenn man obige Regel nicht selbst "erfüllt", erfüllt der Ausführer sie, indem er als ersten Befehl des Konstruktors folgenden Befehl einfügt:

```
super();
```

Wann wird dieser Befehl vom Ausführer akzeptiert bzw. abgelehnt?
(akzeptiert, wenn es in der direkten Oberklasse einen ***Standardkonstruktor*** gibt, sonst abgelehnt)

Zur Erinnerung: Die Hartz-4 Regel für Klassen?
(Wenn der Programmierer in einer Klasse keinen einzigen Konstruktor vereinbart, bekommt die Klasse vom Ausführer einen Standardkonstruktor mit leerem Rumpf "geschenkt")

12.5 Kleinere und größere Klassen und Objekte (S. 300)

Was ist größer: Eine ***Oberklasse*** OK oder eine ***Unterklasse*** UK?
Korrekte Antwort: Tja!

Erläuterung anhand von **S. 301, Bild 12.10**

Wozu Untertypen gut sind

S. 296, Beispiel-02: Eine Reihung mit `E01Punkt`-Objekten ausgeben ("die Horror-Version")

Für diese Methode spricht nur: "Sie läuft aber und produziert die richtigen Ergebnisse".
Das reicht aber bei weitem nicht aus!!

S. 298, Beispiel-03: Eine Reihung mit `E01Punkt`-Objekten ausgeben (so muss man es machen)

Zur Entspannung: **Alan Mathison Turing** (1912-1954), einer der Begründer der Informatik

Bevor man die ersten elektronischen Computer baute, konzipierte und untersuchte der Mathematiker Turing eine Rechenmaschine, die so einfach war, dass niemand an ihrer prinzipiellen Realisierbarkeit zweifelte. Eine solche Turing-Maschine besteht aus einem unbegrenzt langen Band, welches in kleine Abschnitte eingeteilt ist, von denen jeder genau ein Zeichen eines endlichen Alphabets aufnehmen kann. Ein Schreib-Lese-Kopf über dem Band kann bei jedem Arbeitsschritt der Maschine das Zeichen auf dem aktuellen Abschnitt lesen und in Abhängigkeit davon ein bestimmtes Zeichen auf den aktuellen Abschnitt schreiben und einen Abschnitt nach links oder rechts weiterrücken. Ein Programm für eine solche Maschine besteht aus einer endlichen Menge von Befehlen der folgenden Form:

"Wenn das aktuelle Zeichen gleich X ist, dann schreibe Y und gehe einen Abschnitt nach links bzw. nach rechts bzw. bleib wo du bist" (wobei X und Y beliebige Zeichen des Alphabets sind).

Wichtige Erkenntnis 1: Es gibt viele (präzise definierte, mathematische) Probleme, die man mit Hilfe einer solchen Turing-Maschine lösen kann (z. B. das Multiplizieren von dreidimensionalen Matrizen).

Wichtige Erkenntnis 2: Es gibt aber auch (präzise definierte, mathematische) Probleme, die man ***nicht*** mit Hilfe einer solchen Turing-Maschine lösen kann.

Wichtige Vermutung 3: Alle Probleme, die man mit heutigen oder zukünftigen Computern lösen kann, kann man im Prinzip auch mit einer Turing-Maschine lösen.

Im zweiten Weltkrieg arbeitete Turing für die ***Government Code and Cypher School*** in Bletchley Park (d. h. für den britischen Geheimdienst) und half entscheidend dabei, die Maschine zu durchschauen, mit der die deutsche Marine ihre Funksprüche verschlüsselte (die Enigma), und wichtige Funksprüche zu entschlüsseln. Damit hat er vermutlich einer ganzen Reihe von aliierten Soldaten (Engländern, Amerikanern, Franzosen, Russen) das Leben gerettet.

Weil er homosexuell war, wurde Turing nach dem Krieg zu einer Hormonbehandlung "seiner Krankheit" gezwungen, bekam schwere Depressionen und nahm sich das Leben. Inzwischen wurden die entsprechenden Gesetze in England (und ähnliche Gesetze in anderen Ländern) beseitigt. Im September 2009 entschuldigte sich der britische Premierminister Gordon Brown dafür, wie Turing behandelt worden ist.

12. SU Mo 19.12.11

Heute in der Übung: Test 10

12.7 Der Typ und der Zieltyp einer Referenz-Variablen (S. 303)

S. 304, Beispiel-01, Zeile 1:

Von welchem Typ ist die Variable `p01`? (`E01Punkt`)
Von welchem Typ ist ihr Zielwert (das Objekt, auf das sie zeigt)? (`E01Punkt`)

Zeile 5, ebenso.

Zeile 7 bis 11, ebenso.

Merke: Der ***Typ*** einer Variablen ist ***unveränderbar*** (das gilt auch für Referenzvariablen).
Der ***Zieltyp*** einer Referenzvariablen ***kann verändert werden*** (durch eine Zuweisung).

14 Abstrakte Klassen und Schnittstellen (S. 335)

Wiederholung: Eine Reihung, die Objekte verschiedener Typen enthalten kann:

```
E01Punkt[] pr = new E01Punkt[150];
```

Was für Objekte können Komponenten dieser Reihung sein?
(Objekte der Typen `E01Punkt`, `E01Rechteck`, `E01Quadrat`, `E01Ellipse`, `E01Kreis`)

Auf welche Elemente einer Komponenten `pr[i]` können wir zugreifen?
(Nur auf die in der Klasse `E01Punkt` vereinbarten Elemente)

Auf welche Methoden, die in allen Unterklassen von `E01Punkt` vereinbart sind, können wir ***nicht*** zugreifen? (Auf die Methoden `getUmfang` und `getFläche`)

Daraus folgt: `E01Punkt` ist keine gute Wurzelklasse unserer kleinen Klassenhierarchie (die z.B. auf S. 301, Bild 12.10 abgebildet ist).

Eine bessere Wurzelklasse: **S. 336, Beispiel-01**: Die abstrakte Klasse `E02GeoFigur`

Die zwei wichtigsten Eigenschaften einer abstrakten Klasse wie `E02GeoFigur`:

1. Ein Befehl wie new `E02GeoFigur(1.5, 2.5)` ist nicht erlaubt (obwohl die Klasse einen entsprechenden Konstruktor enthält!)

2. Eine abstrakte Klasse darf (ausser Konstruktoren und "konkreten Elementen" auch) ***abstrakte Objektmethoden*** enthalten.

Wie viele abstrakte Objektmethoden enthält die Klassewie `E02GeoFigur` und wie heißen sie?
(2, `getFlaeche`, `getUmfang`)

Merke: Nur ***Objektmethoden*** können ***abstrakt*** sein, es gibt keine abstrakten Klassenmethode;
"***abstrakte Methode***" ist also immer eine Abkürzung für "***abstrakte Objektmethode***".

Regel: Jede konkrete Klasse, die eine ***abstrakte Methode*** erbt, muss sie durch eine ***konkrete Methode*** überschreiben.

S. 337, die Klasse `E02Rechteck` (zum Vergleich: **S. 288**, die Klasse `E01Rechteck`)

Wo in der Klasse `E02Rechteck` werden die von `E02GeoFigur` geerbten abstrakten Methoden überschrieben? (In den Zeilen 51 und 52)

S. 339, Bild 14.1 Eine bessere Klassenhierarchie

Was ist an folgender Reihung `gr` besser als bei `pr`?

```
E02GeoFigur[] gr = E02GeoFigur[150];
```

Der Java-Ausführer ist jetzt sicher, dass jede Komponente `gr[i]` auch Methoden `getUmfang` und `getFläche` hat. Deshalb dürfen wir auf diese Methoden zugreifen.

14.2 Schnittstellen (interfaces) (S. 341)

Eine Schnittstelle hat Ähnlichkeit mit einer Klasse und definiert (wie eine Klasse) einen Typ. Eine Schnittstelle darf aber (etwas vereinfacht gesagt) nur ***abstrakte Methoden*** enthalten.

S. 341, Beispiel-01: Die Schnittstelle `Vergroesserbar`

Wie viele abstrakte Methoden enthält diese Schnittstelle? (2, `verdopple` und `verdreifache`)

S. 342: Schnittstellenregel-1: Eine Klasse darf beliebig viele (0, 1, 2, 3, ...) Schnittstellen implementieren

S. 342, Beispiel-02: Die Klasse `GanzZahl01` implementiert die Schnittstelle `Vergroesserbar`

Folge: Die Objekte dieser Klasse gehören jetzt außer zum Typ `GanzZahl01` auch zum Typ `Vergroesserbar`.

Eine Methode mit einem Vergroesserbar-Parameter:

```
1 static public void versechsfache(Vergroesserbar v) {
2    v.verdopple();
3    v.verdreifache();
4 }
```

Vokabeln und Bezeichnungen:

Eine `Vergroesserbar`-Klasse ist eine Klasse, die
die Schnittstelle `Vergroesserbar` implementiert.
Ein `Vergroesserbar`-Objekt ist ein Objekt einer `Vergroesserbar`-Klasse.

Allgemein: Sei S irgendeine Schnittstelle. Dann gilt:

Eine S-Klasse ist eine Klasse, die die Schnittstelle S implementiert (und nicht etwa ein Mercedes!)
Ein S-Objekt ist ein Objekt einer S-Klasse.

Zur Entsapannung: **Ein bewiesenermaßen unlösbares Problem**

Beispiel für eine D-Gleichung:

$+7*x^1 - 3*x^2 + 5*y^4 - 2*z^5 = 0$

Das Polynom auf der linken Seite der Gleichung darf Potenzen von ***beliebig vielen Variablen*** `x, y, z, x1, x2, ...` aber nur ***ganzzahlige Koeffizienten*** (im Beispiel: +7, -3, +5, -2) enthalten.
Bei D-Gleichungen interessiert man sich nur für ***ganzzahlige Lösungen***.

Beispiele: D-Gleichungen mit und ohne Lösungen

```
 5    x^2 + 2y^2      = 0  // Genau eine Lösung:              x= 0, y=0
 6    x^2 - 4         = 0  // Genau zwei Lösungen:            x= 2 und x=-2
 7    x^1 + 5y^1 - 8  = 0  // unendlich viele Lösungen, z.B. x= 3, y=1
 8    x^2 - y^2 - z^2 = 0  // unendlich viele Lösungen, z.B. x=-5, y=4, z=3
 9    x^2 + 5         = 0  // keine Lösung (leicht zu sehen)
10    x^4 - y^4 - z^4 = 0  // keine Lösung ausser der trivialen(schwer zu beweisen)
```

Schön wäre ein Programm, welches von jeder D-Gleichung feststellen kann, ob sie lösbar ist (d.h. mindestens eine Lösung hat) oder nicht. Leider wurde folgender Satz bereits bewiesen:

Satz: Es gibt keinen Algorithmus, der von jeder D-Gleichung korrekt feststellen kann, ob sie lösbar ist oder nicht.

"D-Gleichungen" werden üblicherweise als "diophantische Gleichungen" bezeichnet, nach dem griechischen Mathematiker ***Diophant von Alexandrien***, der irgendwann zwischen 100 v.Chr. und 350 n.Chr. (vermutlich um 250 n.Chr.) lebte und 13 Bücher über Arithmetik veröffentlichte, von denen 10 bis heute erhalten geblieben sind.

13. SU Mo 02.01.12

Heute in der Übung Test 11

Wiederholung und Fortsetzung: Abstrakte Klassen und Schnittstellen

Was für Dinge darf man innerhalb einer Schnittstelle vereinbaren? (Abstrakte Objektmethoden)

Angenommen, wir vereinbaren eine Klasse `UK` ("unsere Klasse") wie folgt:

```
class UK extends AK implements I1, I2 {
   ...
}
```

Dabei soll `AK` eine abstrakte Klasse sein. Was müssen wir innerhalb von `UK` machen?
(Wir müssen alle ***abstrakten Methoden*** aus `AK, I1` und `I2` durch ***konkrete Methoden*** überschreiben)

Das **Beispiel-01** (auf der nächsten Seite 32) austeilen und besprechen. Dann soll jeder in der Klasse `Schnittstelle01` eine geeignete `hatWert`-Methode ergänzen.

S. 341, Schnittstellenregel-01
S. 343, Schnittstellenregel-02
S. 346, Schnittstellenregel-03

9.4 Klassische Fachbegriffe

Was darf man innerhalb einer Klasse vereinbaren? Wie heißen die Dinge, die man innerhalb einer Klasse vereinbaren darf? (Konstruktoren und Elemente)

Die Elemente, die in einer Klasse vereinbart werden (engl. the members declared in a class),
kann man nach ***drei*** Kriterien in (unterschiedlich viele) Gruppen einteilen:

Kriterium	Gruppen	Anzahl der Gruppen
nach Art	Attribut, Methode, Klasse, Schnittstelle	4
nach Aspekt-zugehörigkeit	Klassenelement (static member), Objektelement (non-static member)	2
nach Erreichbarkeit	öffentlich, geschützt, paketweit-erreichbar, privat	4

Man unterscheidet somit z.B.:

Öffentliche Klassenattribute,
geschützte Objektmethoden,
paketweit-erreichbare Klassen-Klassen (oder: paketweit-erreichbare statische Klassen)
private Objekt-Schnittstellen (oder: private nicht-statische Schnittstellen),
... etc.

Alle möglichen Kombinationen der Gruppeneigenschaften können vorkommen.

Arbeitsblatt, zum Austeilen im 13. SU

Beispiel-01: Eine Methode bearbeitet eine Reihung von Schnittstellen-Variablen

```
 1 ------------------------------------------------------------------
 2 // Datei HatWert.java
 3 interface HatWert {
 4    int wert(); // Diese Methode ist automatisch abstract und public
 5 }
 6 ------------------------------------------------------------------
 7 // Datei K17.java
 8 public class K17 implements HatWert {
 9    int x;
10    int y;
11
12    public int wert() {
13        return 2*x + y;
14     }
15
16     K17(int x, int y) {
17        this.x = x;
18        this.y = y;
19     }
20 }
21 ------------------------------------------------------------------
22 // Datei K18.java
23 public class K18 implements HatWert {
24    int a;
25
26    public int wert() {
27        return a*a + a;
28     }
29
30     K18(int a) {
31        this.a = a;
32     }
33 }
34 ------------------------------------------------------------------
35 // Datei Schnittstelle04.java
36 public class Schnittstelle04 {
37    static public void main(String[] _) {
38       K17[] rk17 = {new K17(2, 5),  new K17(3, 1), new K17(4, 0)};
39       K18[] rk18 = {new K18(3),     new K18(-4),   new K18(2)};
40
41       pln("Der maximale Wert in rk17: " + maxWert(rk17));
42       pln("Der maximale Wert in rk18: " + maxWert(rk18));
43    }
44
45    static int maxWert(K17[] r) {
46       int max = Integer.MIN_VALUE;
47       for (K17 ob : r) {
48          int w = ob.wert();
49          if (w > max) max = w;
50       }
51       return max;
52    }
53
54    ...
55
56    // Eine Methode mit einem kurzen Namen:
57    static void pln(Object ob) {System.out.println(ob);}
58 }
```

Die Methode `maxWert` (ab Zeile 45) kann nur Parameter des Typs `K17[]` bearbeiten. Damit der Aufruf `maxWert(rk18)` (in Zeile 42) korrekt wird, könnten wir eine weitere `maxWert`-Methode mit einem Parameter vom Typ `K18[]` schreiben. Was wäre eine bessere Lösung?

9.5 Objektorientierte Programmierung

Angenommen, wir wollen ein objektorientiertes Programm zur Verwaltung der Beuth Hochschule schreiben? Wie gehen wir vor?

1. Wir fragen: Was sind die wichtigen Objekte in der Beuth Hochschule?

Mögliche Antwort: StudentInnen, Hörsäle, Lehrveranstaltungen, ...

2.1. Wir fragen: Was sind die relevanten Daten einer StudentIn?

Mögliche Antwort: Name, Matrikel-Nr, Noten, ...

3.1. Wir fragen: Was sind die relevanten Aktionen mit Bezug auf eine StudentIn?

Mögliche Antworten: Einschreiben, belegen, eine-Note-bekommen, ...

2.2. Wir fragen: Was sind die relevanten Daten eines Hörsaals? ...
3.2. Wir fragen: Was sind die relevanten Tätigkeiten mit Bezug auf einen Hörsaal? ...

...

4. Wir programmieren eine Klasse Namens `StudentIn` mit

Attributen namens `name`, `matrikel_nr`, `noten`, ... und
Methoden namens `einschreiben`, `belegen`, `bekommtEineNote`, ...

Jedes Objekt dieser Klasse repräsentiert eine StudentIn.

Für Hörsäle, Lehrveranstaltungen etc. programmieren wir entsprechend Klassen namens `Hoersaal`, `Lehrveranstaltung`, ... etc.

Ziel: Die Objekte ***ausserhalb*** von Computern
(StudentInnen, Hörsäle, Lehrveranstaltungen, ...) und
die Objekte ***im Verwaltungscomputer***
(`StudentIn`-Objekte, `Hoersaal`-Objekte, `Lehrveranstaltung`-Objekte, ...)
sollen sich möglichst direkt und offensichtlich entsprechen.

Pakete (die nichts mit Weihnachten zu tun haben)

Motivation: Sie wollen ein großes Java-Programm entwickeln, einige der Klassen aber nicht selbst schreiben, sondern von verschiedenen Firmen dazukaufen. Natürlich hat jede dieser Firmen mindestens eine Klasse namens `Logging` und zwei namens `Test` vereinbart und Sie brauchen alle diese Logging- und Test-Klassen. Wie kann man in solchen Situationen Mehrdeutigkeiten verhindern?

Mit Paketen!

Def.: Ein ***Paket*** ist ein Behälter für Klassen, Schnittstellen und weitere Pakete.

D.h. man kann Pakete schachteln (packages can be nested).

Anmerkung: Ein Paket ist ***kein*** Modul, weil die Bedingung mit dem öffentlichen und dem privaten Teil nicht vollständig erfüllt ist (***geschachtelte Pakete*** sind immer ***öffentlich*** und können nicht als ***privat*** gekennzeichnet werden).

Fachbegriffe: Top-Paket, Name eines Pakets, voller Name eines Pakets, Wald von Paketen.

Paketregel-1: Jede Klasse gehört zu genau ***einem*** Paket. (S. 425)

S. 426, Beispiel-01
S. 426, Beispiel-02

Damit hat auch jede Klasse (nicht nur einen Namen sondern auch) einen **vollen Namen**:
Der volle Name des Paketes gefolgt vom (einfachen) Namen der Klasse.

Das **namenlose Paket**. Wird nur "beim Lernen und Ausprobieren" verwendet. "Ernsthafte Klassen" gehören alle zu einem Paket.

Erreichbarkeit von Klassen in Paketen-mit-Namen

Eine Klasse, die zu einem Paket gehört, ist entweder ***öffentlich*** (`public`) oder ***paketweit erreichbar***.

Anmerkung: Eine innerhalb einer anderen ***Klasse*** vereinbarte Klasse kann
öffentlich, ***geschützt***, ***paketweit erreichbar*** oder ***privat*** sein.

Anmerkung: Die Klassen im namenlosen Paket sind immer nur ***paketweit erreichbar***, egal ob ihre Vereinbarung mit `public` beginnt oder nicht.
Grund: Aus einem Paket-mit-Namen kann man nicht auf Klassen im namenlosen Paket zugreifen,weil:
1. Wenn in einer Klasse K in einem Paket P z.B. der einfache Klassenname `MeineKlasse` steht, dann bezeichet der eine Klasse im Paket P (sonst ist er falsch).
2. `import`-Befehle ohne Paket-Namen, z.B. `import MeineKlasse;`
sind (seit Java 1.4) nicht mehr erlaubt.

Grundlegende Vorstellung: Klassen haben mit ***zwei Baumstrukturen*** zu tun:
1. Dem Baum aller Klassen mit der beerbt-Relation
2. Dem Paket-Wald, in dem jede Klasse zu einem Paket gehört

Wichtig-1: Diese beiden Strukturen sind völlig ***unabhängig voneinander***

Beispiel: Sei `K2` eine Unterklasse von `K1`. Dann kann man `K1` und `K2` in dasselbe Paket tun, oder `K1` in ein "Oberpaket" und `K2` in ein "Unterpaket" tun, oder umgekehrt: `K1` in ein "Unterpaket" und `K2` in ein "Oberpaket", oder `K1` in ein Paket `P1` und `K2` in ein anderes Paket `P2`, wobei die Pakete `P1` und `P2` nichts miteinander zu tun haben.

Wichtig-2: Klassen, die sich in einem gemeinsamen Paket befinden, haben "Privilegien beim Zugriff auf einander", denn für sie sind auch ***paketweit erreichbare*** Elemente und Konstruktoren erreichbar.

S. 434, Bild 17.1: Fünf Klassen in zwei Paketen

K10 ist hier die "Zentralklasse", die anderen stehen "in allen möglichen Verhältnissen" zu K10:

Klasse	K11	K12	K21	K22
im selben Paket wie K10	ja	ja	nein	nein
beerbt K10	ja	nein	ja	nein

In der Klasse K10 sind 4 Elemente vereinbart. Wie heißen diese Elemente und warum heißen sie so?

Dargestellt wird, welche dieser 4 Elemente aus `K10` in den übrigen 4 Klassen erreichbar sind (z.B. ist in `K22` nur noch das Element `publicE` erreichbar.

14. SU Mo 09.01.12

Heute in der Übung Test12

Eine Schwäche der Sprache Java

Mit Hilfe von Paketen kann man innerhalb eines Programms mehrere Klassen verwenden, die den gleichen ***einfachen Namen*** haben(z.B. `Test`). Man muss nur dafür sorgen, dass die ***vollen Namen*** der Klassen sich unterscheiden (z.B. `de.beuth-hochschule.Java.Test` und `de.tu-berlin.Klassen.Test`). Bei Schnittstellen klappt das leider nicht immer.

Zur Erinnerung: Zum ***Profil*** einer Methode gehören der Rückgabetyp, der Methodenname und die Typen der Parameter. Nicht zum Profil gehören die Namen der Parameter und der Rumpf der Methode.

Eine konkrete Klasse `K` kann ***zwei*** abstrakte Methoden mit dem gleichen Profil P erben, z.B. so:

```
class K implements vonfirma01.test01, vonfirma02.test02 { ... }
```

Beide Schnittstellen `test01` und `test02` können z.B. eine abstrakte Methode
`String codeword()` mit dem Profil P gleich `String codeword` enthalten.
Die Klasse `K` kann aber nur ***eine*** konkrete Methode mit dem Profil P vereinbaren. Dass ist inhaltlich nicht angemessen, wenn die Firma01 "etwas ganz anderes von der Methode `codeword` erwartet" als die Firma02. Siehe dazu auch

http://en.wikipedia.org/wiki/Comparison_of_C_Sharp_and_Java#Explicit_interface_implementation

Grabos (graphische Benutzeroberflächen, engl. GUI, graphical user interface)

Eine ***Grabo*** (grafische Benutzeroberfläche) besteht aus ***Grabo-Objekten***. Für jedes Grabo-Objekt ist eine ***grafische Darstellung*** festgelegt (z.B. als Fenster oder als Knopf oder ...) und wenn man ein Grabo-Objekt erzeugt, wird es (mehr oder weniger ***automatisch***) auf dem Bildschirm ***dargestellt***.

Konsolen-Programme und Grabo-Programme

Unterscheiden sich grundlegend voneinander. Charakteristische Eigenschaften:

	Konsolen-Programm	**Grabo-Programm**
Ausführung	***Ein*** Programm, wird ***sequentiell*** ausgeführt	***Mehrere*** Programmteile, die ***nebenläufig*** zueinander ausgeführt werden
Steuerung	Das Programm steuert den Benutzer (befiehlt ihm z.B., Daten einzugeben)	Der Benutzer steuert das Programm (z.B. gibt er Daten ein, "wenn er Lust hat")
Bildschirm	zeichenorientiert (z.B. 25x80 Zeichen)	graphisch (z.B. 1280x1024 Pixel)

S. 531, Beispiel-01: Zwei `JFrame`-Objekte
S. 531, Bild 22.1

Der Programmierer kann Grabo-Klassen als Unterklassen von vorhandenen Grabo-Klassen vereinbaren.

S. 533, Beispiel-02: Die Klasse `Grabo01` erweitert die Klasse `JFrame`
S. 534, Beispiel-03: Das Programm `Grabo01Tst`
S. 535, Bild. 22.2 (Druckfehler korrigieren: "Grabo01-Objekte" statt "JFrame-Objekte")

Plan: Wir wollen eine Fenster-Klasse `Grabo02` mit folgender Eigenschaft schreiben:
Wenn der Benutzer beim Fenster eines `Grabo02`-Objekts den Fenster-Schließen-Knopf ("den mit dem X darin") anklickt, soll eine von uns geschriebene Methode ausgeführt werden (und eine Meldung zur Standardausgabe ausgeben)

Der Name der Methode, die beim Anklicken des Fenster-Schließen-Knopfes ausgeführt wird, ist von der Sprache Java festgelegt: `windowClosing`.

S. 536, Beispiel-04: Die Klasse `Grabo02`

Innerhalb dieser Klasse müssen wir
1. Eine Klasse vereinbaren, die die Klasse `WindowAdapter` erweitert
und darin eine Methode namens `windowClosing` vereinbaren
2. Ein Objekt dieser Klasse erzeugen und
3. bei dem `Grabo02`-Objekt anmelden.

Zur Entspannung: **Englische Vokabeln: boot, bootstrap**

Ein ***boot*** ist ein hoher Schuh oder Schnürstiefel, ein ***bootstrap*** ein Schnürsenkel für einen Schnürstiefel. Ein ***bootstrap*** gilt als simples Werkzeug, welches immer zur Hand ist und mit dem man sich andere Werkzeuge "heranziehen kann", z. B. so: Eine Gruppe von Pfadfindern (natürlich alle in Schnürstiefeln) will ein schweres Stahlkabel über einer Felsspalte anbringen. Dazu knüpfen sie zuerst ihre ***bootstraps*** zusammen und ziehen damit ein dünnes Seil über die Felsspalte. Mit dem dünnen Seil ziehen sie ein dickeres Seil über die Felsspalte und mit dem dickeren Seil schliesslich das Stahlkabel.

Beim Booten eines Rechners liest der Prozessor zuerst von einem bestimmten Gerät einen einzigen Datenblock (z. B. 512 Byte) und springt dann zum ersten Byte dieses Blocks. Der Block sollte ein kleines Ladeprogramm, enthalten, welches ein größeres Ladeprogramm (z. B. ein paar Tausend Bytes) von einer bestimmten Platte liest und zum Anfang dieses Ladeprogramms springt. Das größere Ladeprogramm lädt dann wichtige Teile des Betriebssystems (ein paar Megabyte) und springt an eine Stelle in diesem Betriebssystem.

15. SU Mo 16.01.12

Heute in der Übung Test13 (der letzte!)

Arbeitsblatt **Anonyme Objekte und anonyme Klassen** austeile und besprechen.

Fragen zu einigen Zeilen (des Arbeitsblattes) und **Antworten** dazu:

1 Was wird ausgegeben? `Hallo`
2 Was wird ausgegeben? `Punkt(1.0, 2.0, 3.0)`
3 Welchen Zielwert bekommt die Variable `s`? `[10, 20, 30]`
4 Ein Objekt von welchem Typ wird hier angemeldet? Bei welchem Objekt wird es angemeldet?

Anmerkung: Dieser Befehl steht in einem Konstruktor der Klasse Grabo02.
Angemeldet wird ein ProgTerminator-Objekt, beim aktuellen Objekt this, welches vom Konstruktor gerade initialisiert wird

5 Nach welchem Bauplan wird das Objekt, welches hier mit `new` erzeugt wird, gebaut?
Wodurch unterscheidet sich dieses Objekt von einem `Punkt3D`-Objekt?

Nach einem Bauplan, der einer (anonymen) Unterklasse von Punkt3D entspricht.
Das Objekt hat eine andere toString-Methode als ein Punkt3D-Objekt.

11 Nach welchem Bauplan wird das Objekt, welches hier nit `new` erzeugt wird, gebaut?
Wodurch unterscheidet sich dieses Objekt von einem `WindowAdapter`-Objekt?

Nach einem Bauplan, der einer (anonymen) Unterklasse von WindowAdapter entspricht.
Das Objekt hat eine andere windowClosing-Methode als ein WindowAdapter-Objekt.

18 Was ist ungewöhnlich am `new`-Befehl in dieser Zeile?

Der Befehl hinter new sieht aus wie der Aufruf eines Konstruktors der Schnittstelle ActionListener, obwohl Schnittstellen keine Konstruktoren haben.

24 Was wird ausgegeben? Was hat diese Ausgabe mit der Variablen `pEn` (siehe Zeile 5) zu tun?

Ausgegeben wird Point(10, 20, 30). Das anonyme Objekt, welches als Parameter von pln erzeugt wird, stimmt genau mit dem Objekt pEn überein ("wie ein Zwilling").

Plan

Am Mo 23.01.2012 wird im SU die Klausur vorbereitet. In der Übung kann man sich schon auf die Klausur vorbereiten.

Am Mo 30.01.2012 findet nur dann etwas statt (SU und/oder Ü), wenn ich spätestens 3 Tage vorher eine Email bekommen habe, in der mindestens 3 Teilnehmer fest zusagen, zu kommen.

Am Mi 01.02.2012 schreiben wir (ab 18 Uhr, im Beuth-Saal) die Klausur.

Rückgabe der Klausur: Mo 06.02.2012, im 3. Block, Raum B345 ("wie immer")

22.4 Aktionen, Ereignisse, ...

Der ***Benutzer*** eines Grabo-Programms kann ***Aktionen*** ausführen (z.B. die Maus über einem Fenster bewegen, auf einen Knopf klicken, auf eine Taste der Tastatur drücken etc.)

Eine solche Aktion bewirkt, dass das betreffende Grabo-Objekt
(z.B. das `JFrame`-Objekt oder das `JCheckBox`-Objekt, mit dem der Benutzer gerade interagiert hat)
ein ***Ereignis-Objekt*** erzeugt
(z.B. ein `WindowEvent`-Objekt oder ein `ActionEvent`-Objekt)
und, mit diesem Ereignis-Objekt als Parameter, bestimmte Behandler-Methoden aufruft.

Die Behandler-Methoden müssen sich in Listener-Objekten befinden
(z.B. in `WindowListener`-Objekten oder in `ActionListener`-Objekten)
welche vorher bei dem betreffenden Grabo-Objekt angemeldet wurden.

Wie kann man herausfinden, ***was für Ereignisse*** die Objekte einer Grabo-Klasse GK erzeugen können?

1. Wir ermitteln alle `addXXXListener`-Methoden der Klasse GK.

Beispiele: Für GK gleich `JFrame`, `JButton` und `JLabel` sind das die folgenden Methoden:

Klasse JFrame	Klasse JButton	Klasse JLabel
addComponentListener addFocusListener addHierarchyBoundsListener addHierarchyListener addInputMethodListener addKeyListener addMouseListener addMouseMotionListener addMouseWheelListener addContainerListener addPropertyChangeListener addPropertyChangeListener addWindowFocusListener addWindowListener addWindowStateListener	addComponentListener addFocusListener addHierarchyBoundsListener addHierarchyListener addInputMethodListener addKeyListener addMouseListener addMouseMotionListener addMouseWheelListener addContainerListener addPropertyChangeListener addPropertyChangeListener addAncestorListener addVetoableChangeListener addActionListener addChangeListener addItemListener	addComponentListener addFocusListener addHierarchyBoundsListener addHierarchyListener addInputMethodListener addKeyListener addMouseListener addMouseMotionListener addMouseWheelListener addContainerListener addPropertyChangeListener addPropertyChangeListener addAncestorListener addVetoableChangeListener

2. Wenn man vom Namen einer solchen Methode den Anfang `add` wegläßt, erhält man
den Namen einer sog. ***Listener-Schnittstelle*** (z.B. wird aus dem Methoden-Namen
`addComponentListener` der Schnittstellen-Name `ComponentListener`)
Jede solche Listener-Schnittstelle entspricht einer ***Oberart von Ereignissen***,
die von GK-Objekten erzeugt werden können

3. Jede Methode in einer solchen Listener-Schnittstelle entspricht ***einer Art von Ereignissen***,
die von GK-Objekten erzeugt werden können.

S. 546, Def.: ***Ereignis***

S. 547, ***Arten*** von Ereignissen und ***Oberarten*** von Ereignissen

Arbeitsblatt für den 15. SU

Anonyme Objekte und anonyme Klassen

Beispiele für *anonyme Objekte*:

```
 1    pln(new String("Hallo"));
 2    pln(new Punkt3D(1.0, 2.0, 3.0));
 3    String s = Arrays.toString(new int[]{10, 20, 30});
 4    this.addWindowListener(new ProgTerminator()); // siehe Buch S. 536, Z. 63
```

Beispiele für benannte Objekte *anonymer Klassen*:

```
 5    Punkt3D pEn = new Punkt3D(1.0, 2.0, 3.0){ // pEn wie "Punkt auf Englisch"
 6       public String toString() {
 7          return super.toString().replace("Punkt", "Point");
 8       }
 9    };
10
11    static WindowAdapter arnold = new WindowAdapter() {
12       public void windowClosing(WindowEvent we) {
13          pln("Feierabend!")
14          System.exit(0);
15       }
16    };
17
18    static ActionListener alfred = new ActionListener() {
19       public void actionPerformed(ActionEvent ae) {
20          pln("Hab grad keine Lust!");
21       }
22    }
23
```

Beispiel für *ein anonymes Objekt einer anonymen Klasse*:

```
24    pln(new Punkt3D(1.0, 2.0, 3.0){
25       public String toString() {
26          return super.toString().replace("Punkt", "Point");
27       }
28    });
```

Fragen zu einigen Zeilen:

1 Was wird ausgegeben?
2 Was wird ausgegeben?
3 Welchen Zielwert bekommt die Variable `s`?
4 Ein Objekt von welchem Typ wird hier angemeldet? Bei welchem Objekt wird es angemeldet?

5 Nach welchem Bauplan wird das Objekt, welches hier mit `new` erzeugt wird, gebaut?
Wodurch unterscheidet sich dieses Objekt von einem `Punkt3D`-Objekt?

11 Nach welchem Bauplan wird das Objekt, welches hier nit `new` erzeugt wird, gebaut?
Wodurch unterscheidet sich dieses Objekt von einem `WindowAdapter`-Objekt?

18 Was ist ungewöhnlich am `new`-Befehl in dieser Zeile?

24 Was wird ausgegeben? Was hat diese Ausgabe mit der Variablen `pEn` (siehe Zeile 5) zu tun?

Anmerkung:

In einem Java-Quellprogramm ist die Zeichenfolge `});` ist ein typisches Kennzeichen für
die Benutzung eines ***anonymen Objekts*** einer ***anonyomen Klasse***.

Zur Entspannung: **Kurt Gödel** (1906-1978, Österreich-Ungarn, Princeton, USA)

Studierte Physik und Mathe in Wien, frühe Begabung für Mathe.

1931: "Über formal unentscheidbare Sätze der Principia Mathematica und verwandte Sätze". Die Principia Mathematika (3 Bände, erschienen 1910-1913) war ein philosophisch-mathematisch wichtiges Werk von Bertrand Russell (1872-1970) und Alfred North Whitehead (1861-1947).

Mit diesem Papier zerstöre Gödel die Hoffnung des Mathematikers David Hilbert (1862-1943), alle mathematischen Sätze rein formal aus einer Basis von Axiomen abzuleiten.

1932: Habilitation in Wien.
1933: Hitler kam an die Macht.
1934: Vorlesungen in Princeton.
1938: "Anschluss" Österreichs an Deutschland.
1940: Auswanderung in die USA, bis 1978 in Princeton, Freund von Einstein. "Consistency of the Axiom of Choice and the Generalized Continuum Hypothesis with the Axioms of Set Theory".

Einer der bedeutendsten Mathematiker des 20. Jahrhunderts. Starb in einer Nervenheilanstalt an Unterernährung, weil er Angst vor einer Vergiftung hatte.

16. SU Mo 23.01.12

Heute kein Test.

Vorbereitung auf die Klausur (am Mi 01.02.2012, ab 18 Uhr, im Beuth-Saal)

1. Erlaubte Unterlagen: 5 Blätter (max. Größe: DIN A 4). Ob oder wie die Blätter beschriftet sind, können Sie frei wählen.

2. Tip: Während der Klausur erstmal alle Aufgaben "überfliegen", dann die leichteste Aufgabe bearbeiten, danach die zweitleichteste, ..., zuletzt die schwerste.

3. Schreiben Sie jede Ihrer Lösungen auf die Vorderseite eines neuen Blattes. Lassen Sie die Rückseite Ihrer Lösungsblätter grundsätzlich leer.

4. Sie dürfen mit Bleistift schreiben.

Jetzt noch ein paar Bemerkungen zum Inhalt der Klausur

1. Sie sollten Ganzzahlen in jedes beliebige Zahlensystem umwandeln können (wie in Aufgabe 1) und mit Java-Befehlen auf die einzelnen Ziffern einer Zahl zugreifen können (wie in den Aufgaben 2 und 5).

2. Beim Schreiben von ***Funktionen*** sollten Sie, wenn es möglich ist, die Regel "Einfache Fälle zuerst" anwenden können und anwenden.

Beispiel-01:

```
 1 public static boolean istPrim(int n) {
 2    // Verlaesst sich darauf, dass n nicht negativ ist.
 3    // Liefert true genau dann wenn n eine Primzahl ist.
 4
 5    // Einfache Fälle:
 6    if (n <= 1) return false;
 7    if (n == 2) return true;
 8    if (n%2 == 0) return false;
 9
10    // Kompliziertere Fälle:
11    ...
```

Beispiel-02:

```
12 public static int hoch(int b, int exp) {
13    // Verlaesst sich darauf, dass b und exp nicht negativ sind.
14    // Liefert den Wert von "b hoch exp".
15
16    // Einfache Fälle:
17    if (b   <= 1) return b;
18    if (exp == 0) return 1;
19    if (exp == 1) return b;
20
21    // Kompliziertere Fälle:
22    ...
```

3. Beim Schreiben von `if`-Befehlen sollten ebenfalls immer die Regel "Einfache Fälle zuerst" anwenden können und anwenden.

Beispiel-03: Ein schlechter `if`-Befehl (wie man es nicht machen soll):

```
1    if (r >= 0) {
2       for (int i=0; i<r.length; i++) {
3          if (r17[i] == n) return true;
4       }
5       return false;
6    } else {
7       return true;
8    }
```

Beispiel-04: Eine bessere Variante (wie man es machen soll):

```
1    if (r < 0) return false;
2    for (int m : r) {
3       if (m == n) return true;
4    }
5    return false;
```

4 Wie kann man (in einem Java-Programm) feststellen, ob ein String `s11` kleiner bzw. größer als ein String `s12` ist?

S. 227, Beispiel-03

5. Ein String-Literal verhält sich wie ein Name für ein String-Objekt

S. 229, Bild 10.2
S. 230, Beispiel-02

6. Objekte einer Klasse als Bojen darstellen und sie dadurch verstehen

```
 1 class ZweiTupel {
 2    static int anz = 0;
 3    int x;
 4    int y;
 5
 6    public ZweiTupel(int x, int y) {
 7       this.x = x;
 8       this.y = y;
 9       anz++;
10    }
11    static public void main(String[] _) {
12       ZweiTupel anna = new ZweiTupel(10, 20);
13       ZweiTupel bert = new ZweiTupel(30, 40);
14       ZweiTupel carl = new ZweiTupel(50, 60);
15       ...
16 }
```

6.1 Stellen Sie die Variablen `anna` und `bert` als Bojen dar.

6.2. Wie viele Module gibt es, wenn der Ausführer die Zeile 12 fertig ausgeführt hat?
6.3. Wie viele `int`-Variablen gibt es in diesem Moment und wie heißen diese `int`-Variablen mit vollen Namen?

6.4. Wie viele Module gibt es, wenn der Ausführer die Zeile 14 fertig ausgeführt hat?
6.5. Wie viele `int`-Variablen gibt es in diesem Moment und wie heißen diese `int`-Variablen mit vollen Namen?

Einfache und volle Namen von Variablen:

Einfacher Name	Voller Name
anz	ZweiTupel.anz
x	anna.x, bert.x, carl.x
y	anna.y, bert.y, carl.y
anna	anna
bert	bert
carl	carl

7. Sie sollten kurze Folgen von Java-Befehlen mit Papier und Bleistift ausführen können (siehe Kapitel 3 im Buch). Dabei sollten Sie insbesondere wissen, wann und wie man Variablen erzeugt und wann man sie wieder zerstört. Beispiel für entsprechende Aufgaben (mit Lösungen) findet man in fast allen früheren Klausuren.

8. Sie sollten Fragen zum Stoff der Lehrveranstaltung TB3-IN3 (z.B. solche, die in den Tests vorkamen und ähnliche) beantworten können. Ihre Antworten sollten möglichst kurz sein und die richtigen Fachbegriffe enthalten.

Beispiele für Fragen (und Antworten):

8.1. Was ist eine Klasse? (Ein Modul und ein Bauplan für Module)
8.2. Was macht (oder: bewirkt, leistet) eine Prozedur? (Sie verändert den Inhalt von Wertebehältern)
8.3. Was macht (oder: bewirkt, leistet) eine Funktion? (Sie berechnet und liefert einen Wert)
8.4. Objekte welcher Klassen stellen Zeichenketten dar? (String, StringBuilder)
8.5. Was ist eine Variable? (Ein Wertebehälter, dessen Inhalt man beliebig oft verändern kann)
8.6. Was ist ein Programm? (Eine Folge von Befehlen, die von einem Programmierer geschrieben wurde und von einem Ausführer ausgeführt werden kann)
8.7. Was ist ein Modul? (Ein Behälter für Variablen, Unterprogramme, Typen etc., der aus mind. 2 Teilen besteht, einem öffentlichen und einem privaten Teil).

Zur Entspannung: **Gottfried Wilhelm Leibniz** (1646-1716, Leipzig-Hannover)

Philosoph, Politiker, Forscher. Erfand (zeitgleich und unabhängig von Isaac Newton, 1643-1727) die Differentialrechnung. Konzipierte einen Logikkalkül und betrieb Wahrscheinlichkeitsrechnung (mit Anwendung z. B. auf Würfelspiele).

Führte 1673 der Royal Society in London eine Rechenmaschine vor. Heute ist nicht ganz klar, was diese Maschine praktisch konnte und was nicht. 1894 wurde die Maschine von Leibniz an der TU Dresden restauriert. 19?? fand ein Prof. Lehmann an der TU Dresden heraus, dass die Restaurierungsarbeiten auf einem Denkfehler beruhten und das Funktionieren der Maschine verhinderten. Prof. Lehmann baute dann nach den Plänen von Leibniz eine funktionierende Maschine.

Leibniz beschrieb als erster das binäre Zahlensystem.

Nächster Termin: Am kommenden **Mo 30.01.12** wird (wie gewohnt) eine Vorlesung stattfinden (im 3. Block, im gewohnten Raum B345). Dabei werden vor allem Fragen der TeilnehmerInnen behandelt und von ihnen gewünschte Themen wiederholt.

Das wars wohl für dieses Semester. Viel Erfolg bei der Vorbereitung auf die Klausur.