Ein paar Vokabeln einer Fachsprache

Programmierer: *Schreibt* Programme und *übergibt* sie dem Ausführer.

Ausführer: Ein Wesen oder Ding, welches Programme (die der Programmierer ihm übergibt) *prüft, akzeptiert* oder *ablehnt* und (wenn der *Benutzer* es ihm befiehlt) *ausführt*. Beispiele für Java-Ausführer:

- Ein Java-kundiger Mensch mit Papier, Bleistift und Radiergummi.
- Ein PC unter Windows mit dem Java-Compiler `javac.exe` und dem Java-Interpreter `java.exe` der Firma Sun/Oracle.
- Ein PC unter Linux mit dem Java-Compiler `gcj` und dem Java-Interpreter `gij` von Gnu.
- Ein PC unter Windows mit dem Java-Quellcode-Interpreter `drjava` von der Rice University.

Benutzer: *Befiehlt* dem Ausführer, ein Programm *auszuführen.* Muss dem Ausführer alle benötigten Eingabedaten zur Verfügung stellen und bekommt von ihm alle Ausgabedaten des Programms.

Programm: Eine *Folge* von *Befehlen*, die ein Programmierer geschrieben hat und die ein Ausführer ausführen kann.

Vereinbarung (declaration): Ein *Befehl* (des Programmierers an den Ausführer), etwas zu erzeugen (z.B. eine *Variable* oder ein *Unterprogramm* oder einen *Typ* oder eine *Klasse* oder ...).
Achtung: *Werte* können nicht *vereinbart* oder *erzeugt* werden, sondern werden *berechnet* (siehe unten).

Ausdruck (expression): Ein *Befehl* (des Programmierers an den Ausführer) einen *Wert* zu *berechnen.*

Anweisung (statement): Ein *Befehl* (des Programmierers an den Ausführer), den Inhalt bestimmter *Wertebehälter* zu *verändern.*

Wertebehälter: Eine *Variable* oder
eine *Ein-/Ausgabe-Einheit* wie z.B. ein Bildschirm, eine Tastatur, ein Drucker, eine Datei, ...

Einfache Anweisung (simple statement): Eine *Anweisung*, die keine Anweisungen enthält (z.B. die Zuweisungs-Anweisung oder die `return`-Anweisung).

Zusammengesetzte Anweisung (compound statement): Eine *Anweisung*, die Anweisungen enthalten kann (z.B. die `if`-Anweisung, die `switch`-Anweisung, die `for`-Anweisung ...).

Einfacher Ausdruck (simple expression): Ein *Ausdruck*, der keine Ausdrücke enthält. Ein einfacher Ausdruck ist entweder ein Literal (wie z.B. `17` oder `2.5` oder `'A'` oder `"Hallo"`) oder ein Variablen-Name (wie z.B. `x` oder `summe_25`).

Zusammengesetzter Ausdruck (compound expression): Ein *Ausdruck*, der Ausdrücke enthält.
Beispiele: Der Ausdruck `17 + x` enthält die Ausdrücke `17` und `x`.
Der Ausdruck `y * 17 + z` enthält die Ausdrücke `y * 17` und `z`.

Unterprogramm (subroutine): Eine *Folge von Befehlen*, die der Programmierer zusammengefasst und mit einem *Namen* und (0 oder mehr) *formalen Parametern* versehen hat. Ein Unterprogramm kann man *aufrufen* (Klassen, Typen, Variablen etc. kann man nicht aufrufen.).

Methode (method): Ein in einer Klasse vereinbartes *Unterprogramm*.

Attribut (field): Eine (direkt) in einer Klasse vereinbarte *Variable*.

Lokale Variable (local variable): Eine in einem *Unterprogramm* vereinbarte Variable.

Anmerkung: In Java gilt: Jedes *Unterprogramm* muss in einer Klasse vereinbart werden und ist somit eine *Methode*. Eine Variable muss in einer Klasse (als Attribut) oder in einer Methode (als lokale Variable) vereinbart werden. In C++ gibt es zusätzlich Unterprogramme, die außerhalb einer Klasse vereinbart wurden (also keine Methoden sind), und Variablen, die außerhalb von Klassen und außerhalb von Unterprogrammen vereinbart wurden (also keine Attribute und keine lokalen Variablen sind).

Prozedur (procedure): Ein Unterprogramm, das zum *Verändern der Inhalte von Wertebehältern* dient. Jeder Aufruf einer Prozedur ist eine *Anweisung*. Ein Prozeduraufruf liefert keinen Wert.

Funktion (function): Ein Unterprogramm, das zum *Berechnen eines Wertes* dient.
Jeder Aufruf einer Funktion ist ein *Ausdruck*. Ein Funktionsaufruf liefert (yields or returns) einen Wert.

Typ: Ein Bauplan für Variablen.

Primitiver Typ (Java-spezifisch): Einer der 8 Typen
`byte, char, short, int, long, float, double, boolean.`

Referenztyp (Java-spezifisch): Alle nicht-primitiven Typen, z.B. `Integer, JFrame, String, String[], String[][][], ArrayList<String>, ArrayList<ArrayList<String>>, Collection<String>`, ...

Reihungstyp (array type): Zu jedem (Komponenten-) Typ namens `ABC` gibt es in Java einen *Reihungstyp* namens `ABC[]` (Reihung von `ABC`-Variablen). Beispiele:

Komponententyp	**Reihungstyp**	**Der Reihungstyp wird gesprochen:**
`int`	`int[]`	Reihung von int-Variablen
`String`	`String[]`	Reihung von String-Variablen
`int[]`	`int[][]`	Reihung von Reihungen von int-Variablen

Reihung (array): Ein Objekt eines Reihungstyps.
Eine Reihung vom Typ `ABC[]` hat *Komponenten* (elements) vom Typ `ABC`.

Sammlungstyp (collection type): (Java-spezifisch) Eine Klasse, die die Schnittstelle `java.util.Collection` implementiert. Beispiele:

Komponententyp	**Sammlungstyp**	**wird gesprochen:**
`String`	`ArrayList<String>`	ArrayList von String-Variablen
`String`	`HashSet<String>`	HashSet von String-Variablen
`HashSet<String>`	`ArrayList<HashSet<String>>`	ArrayList von HashSet von String-Variablen

Als *Komponententyp* eines Sammlungstyps sind in Java nur *Referenztypen* erlaubt
(die 8 primitiven Typen sind *nicht* erlaubt, wohl aber ihre **Hüllklassen**, siehe unten).

Sammlung: Ein *Objekt* eines Sammlungstyps.
In einer Sammlung kann man Objekte *sammeln* (d.h. man kann Objekte in die Sammlung einfügen, sie dort suchen und aus der Sammlung entfernen). Eine Sammlung vom `Typ ArrayList<ABC>` (oder vom Typ `HashSet<ABC>` oder ...) hat *Komponenten* (elements) vom Typ `ABC`.

Grabo-Klasse (GUI class) (Java-spezifisch): Eine Unterklasse der Klasse `java.awt.Component`.
Beispiele: `JFrame, Window, JButton, Button, JLable, Label.`

Grabo-Objekt (GUI object): Ein Objekt einer Grabo-Klasse. Für jedes Grabo-Objekt ist eine graphische Darstellung definiert. Wenn man ein Grabo-Objekt erzeugt, erscheint diese Darstellung (mehr oder weniger) automatisch auf dem Bildschirm.

Behälterklasse (container class) (Java-spezifisch): Eine Unterklasse der Grabo-Klasse `java.awt.Container`.
Beispiele: `JFrame, Window, JButton.`

Behälter (container): Ein Objekt einer Behälterklasse.
In einen Behälter kann man Grabo-Objekte einfügen und daraus wieder entfernen.
Ein Grabo-Objekt kann sich zu jedem Zeitpunkt in höchstens einem Behälter befinden (im Gegensatz dazu kann ein Objekt sich gleichzeitig in mehreren Sammlungen befinden).

Ausnahmeklasse (exception class) (Java-spezifisch): Eine Unterklasse der Klasse `java.lang.Throwable`.

Ausnahme (exception): Ein *Objekt* einer Ausnahmeklasse.
Eine Ausnahme kann *geworfen* werden (mit einer `throw`-Anweisung) und kann *gefangen und behandelt* werden (mit einer `try-catch`-Anweisung).

Primitive Variable: Eine Variable eines *primitiven Typs*. Besteht aus 2 bis 3 Teilen:
Einer *Referenz* und einem *Wert* (müssen sein) und einem *Namen* (kann sein).

Referenzvariable: Eine Variable eines *Referenztyps*. Besteht aus 2 bis 4 Teilen:
Einer *Referenz* und einem *Wert* (müssen sein) und einem *Namen* (kann sein)
und einem *Zielwert* (kann sein).
Wenn eine Referenzvariable den Wert `null` (sprich: nall) hat, dann hat sie *keinen* Zielwert.
Ist ihr Wert ungleich null, dann hat sie ("garantiert") einen Zielwert.

Modul: Ein Behälter für *Variablen*, *Unterprogramme*, *Typen*, *Module*, ... etc., der
aus mindestens 2 Teilen besteht, einem *öffentlichen Teil* und einem *privaten Teil*.
Auf die Größen im privaten Teil kann man von außerhalb des Moduls *nicht* zugreifen.

Klasse (class): Ein *Modul* und gleichzeitig ein *Bauplan für Module*. Die nach einem solchen Bauplan gebauten Module werden *Objekte* (oder: *Instanzen*) der Klasse genannt.

Element einer Klasse (member of a class): Eine Größe, die innerhalb einer Klassenvereinbarung vereinbart wurde und kein Konstruktor ist. Ein Element (member) ist entweder ein *Attribut* (field) oder eine *Methode* (method) oder eine *Klasse* (class) oder eine *Schnittstelle* (interface).

Konstruktor (constructor): Hat Ähnlichkeit mit einer *Methode*, gilt aber nicht als *Element* (member) der betreffenden Klasse und wird ohne Rückgabetyp vereinbart. Ein Konstruktor kann 0 oder mehr Parameter haben und muss genau so heißen, wie die Klasse, zu der er gehört. Konstruktoren dienen dazu, mit `new` erzeugte Objekte zu *initialisieren* (sie dienen nicht dazu, etwas zu "konstruieren").

Standardkonstruktor (standard constructor): Konstruktor mit 0 Parametern.
Der Rumpf eines Standardkonstruktors kann leer oder nicht-leer sein.

Hüllklasse (wrapper class): Zu jedem der 8 Primitiven Typen gibt es eine zugehörige Hüllklasse.
Einen Wert des primitiven Typs kann man in ein Objekt der betreffenden Hüllklasse *einhüllen*.

Paket (package): Ein Behälter für Klassen, Schnittstellen und Pakete.

Top-Paket (top package): Ein Paket, welches in keinem anderen Paket enthalten ist.

Generische Klasse (generic class): Eine Klasse mit einem oder mehreren Typ-Parametern. Beispiel:

```
class Karl<A, B> { ... }
```

Eine *nicht-generische* Klasse definiert genau *einen* Typ. Eine *generische* Klasse wie `Karl` definiert *viele* Typen, z.B. `Karl<String, String>`, `Karl<Integer, String>`, `Karl<String, Integer>`, ...
Alle Objektmethoden (non-static methods) einer generischen Klasse sind *generische Methoden*. Die Klassenmethoden (static methods) einer generischen Klasse sind nicht-generische Methoden, es sei denn, sie haben eigene Typ-Parameter (siehe **Generische Methode**).

Generische Methode (generic method): Eine *Objektmethode* (non-static method) einer generischen Klasse oder eine Methode mit eigenen Typ-Parametern. Beispiel für letzteres:

```
static <A, B> void machWas(int n, A a, B b) { ... }
```

Diese Methodenvereinbarung definiert viele Klassen-Methoden (**static methods**), z.B.

```
static void machWas(int n, String  a, String  b) { ... }
static void machWas(int n, Integer a, String  b) { ... }
static void machWas(int n, String  a, Integer b) { ... }
...
```

Syntaktische Größe: Eine Zeichenkette, welche Teil eines Quellprogramms sein kann, z.B.
ein *Literal* wie `17` oder `2.5` oder `'A'` oder `"Hallo!"`
ein *Variablen-Name* wie `x` oder `summe_25`
eine *Variablen-Vereinbarung* wie `int otto;` oder `String emil = "Hallo!";`
eine *Methoden-Vereinbarung* wie `int plus1(int n) {return n+1;}`
etc.

Semantische Größe: Ein Ding, welches während der Ausführung eines Programms vom Ausführer *erzeugt* oder *berechnet* wird, z.B. eine *Variable* (wird erzeugt), eine *Methode* (wird erzeugt), ein *Wert* (wird berechnet), eine *Klasse* (wird erzeugt), ein *Objekt* (wird erzeugt), ...