Notiz

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Thema:	Programmformatierungen und Glossar zur Veranstaltung „Grundlagen der Programmierung“
Autoren:	Prof. Dr. Stephan Kleuker
Version / Datum:	1.5 / 27.09.2023

Die folgenden Richtlinien zur Programmierung gelten für die Veranstaltung „Grundlagen der Programmierung“, sie sind angelehnt an reale Vorgaben in Unternehmen und insbesondere die Vorgaben von Sun (Oracle, http://www.oracle.com/technetwork/java/codeconventions-150003.pdf, auch https://www.securecoding.cert.org/confluence/display/java/Java+Coding+Guidelines, auch https://google.github.io/styleguide/javaguide.html auch https://github.com/twitter/commons/blob/master/src/java/com/twitter/common/styleguide.md). Generell gilt für alle Projekte, dass die entwickelnden Personen die Vorgaben einzuhalten haben, damit die Programme langfristig für andere lesbar sind, meist ist eine werkzeuggestützte Überprüfung möglich.

Einige der folgenden Regeln sind Java-spezifisch, andere gelten auch für andere Programmiersprachen.

- alle verwendeten Namen sind sprechend, d. h. sie enthalten mindestens ein Nomen oder ein Adjektiv; Abkürzungen dürfen nur verwendet werden, wenn Sie aus dem Fachgebiet des Programms stammen (z.B. kg statt kilogramm)
- alle verwendeten Namen werden korrekt und möglichst in einer einheitlichen Sprache (deutsch oder englisch) geschrieben
- Ausnahmen von der vorherigen Regel sind nur für Variablen mit sehr kurzer Lebensspanne möglich, dies können Schleifenzähler oder lokale Variablen sein
- Klassennamen werden groß geschrieben (Linie, nicht linie), auch Interface-Namen
- Klassennamen werden als Einzahl geschrieben (Linie, nicht Linien)
- Objektvariablen (auch Exemplarvariablen genannt) werden klein geschrieben (startpunkt, nicht Startpunkt)
- Objektvariablen, die nur einzelne Werte beschreiben, stehen in Einzahl (startpunkt, nicht startpunkte)
- Objektvariablen, die mehrere Werte (also vom Typ Sammlung, z. B. ArrayList, sind) enthalten, stehen in der Mehrzahl (qualifikationen, nicht qualifikation)
- bei allen aus mehreren Wörtern zusammengesetzten Namen werden ab dem zweiten Wort die ersten Buchstaben jeweils großgeschrieben (eingeschriebenAm, nicht eingeschriebenam oder eingeschrieben_am); die gilt nicht für zusammengesetzte Wörter (startpunkt, nicht startPunkt) (-> auch Kamelhöckernotation und Binnenmajuskeln genannt)
- Konstanten werden in ausschließlich großen Buchstaben geschrieben (PI, DATEINAME); dies sind auch die einzigen Namen, in denen Unterstriche in Namen erlaubt sind (MAX_VALUE)
- Vor jeder Nutzung einer Objektvariablen in einer Methode steht das Schlüsselwort this, also immer this.objektvariable

- Vor jeder Nutzung einer Klassenvariablen oder Klassenmethode steht immer der Name der Klasse (Math.abs(-42))
- Der break-Befehl wird ausschließlich in switch-case-Befehlen genutzt
- Der continue-Befehl wird nie genutzt

- Jede Variable wird in einer eigenen Zeile deklariert.

richtig

```
Punkt startpunkt;
Punkt endpunkt;
```

falsch

```
Punkt startpunkt, endpunkt;
```

- Blockklammern bei Methoden, Alternativen und Schleifen werden in einem einheitlichen Format gesetzt; hierbei ist die BlueJ-Variante oder die folgende etwas kompaktere Version möglich.

richtig (nach Sun)

```
public boolean hatHoffnung() {
  return true;
}
```

auch richtig (nach BlueJ)

```
public boolean hatHoffnung()
{
  return true;
}
```

- Jede Programmzeile ist durch eine Einrückung eindeutig den umgebenden Block zugeordnet, die Einrückung erfolgt einheitlich (zwei oder vier Leerzeichen, kein Tabulator).

richtig

```
public boolean hatHoffnung() {
  int alter = this.alterBerechnen();
  if (alter > 30) {
    return false;
  }
  else {
    return true;
  }
}
```

falsch

```
public boolean hatHoffnung(){
int alter = this.alterBerechnen();
if (alter > 30){
return false;
}
else {
return true;
}
}
```

- Jeder für den Ablauf eines Programms wichtige Befehl steht in einer eigenen Zeile. Dies erhöht die Lesbarkeit und macht Messungen zu Testüberdeckungen eindeutig.

richtig

```
if (this.startpunkt.equals(this.endpunkt)){
  return false;
}
```

falsch

```
if (this.startpunkt.equals(this.endpunkt)){  return false; }
```

- Jeder Programmblock in einem if oder in Schleifen wird mit geschweiften Klammern umrahmt, auch wenn es nur eine Zeile ist.

  richtig

  ```
  if (this.startpunkt.equals(this.endpunkt)){
    return false;
  }
  ```

  falsch

  ```
  if (this.startpunkt.equals(this.endpunkt))
    return false;
  ```

- Jede Programmzeile ist maximal 70 Zeichen lang, wird diese Länge überschritten, wird im Programmcode ein Zeilenumbruch eingebaut, dabei wird eine Einrückung genutzt, die doppelt so groß wie die normale Einrückung ist. Parameterlisten werden unmittelbar vor einem Komma umgebrochen, längere Methodenaufrufe vor dem Punkt zur Wahl der neuen Methode (anders als im Sun-Standard von 1997, da dieser Ansatz erst später verbreitet wurde). Bei Booleschen Ausdrücken wird der Verknüpfungsoperator in die neue Zeile übernommen, weiterhin ist es sinnvoll, dass der Aufbau einer Klammerstruktur durch Einrückungen leichter erkennbar wird. Folgendes Beispiel zeigt die genannten Umbrüche, allerdings für noch kürzere Zeilen. Durch die ersten Zeilen der jeweiligen Zeile wird so bereits deutlich, dass dies eine Fortsetzung eines Befehls ist.

  ```
  public int einZiemlichLangerMethodenname(int einErsterParameter
      , String einHoffentlichWichtigerText) {
    int zwischenergebnis = this.nochEineMethode(this.objektvariable
      , 2 * einErsterParameter);
    if(  (   !zwischenergebnis.equals(einErsterParameter – 12345)
          || !zwischenergebnis.equals(einErsterParameter – 123456) )
        && einErsterParameter > 42){
      return zwischenergebnis;
    }
    return 42;
  }
  ```

- Bei import-Anweisungen wird immer jede genutzte Klasse außerhalb von java.lang einzeln angegeben. Die unsaubere Abkürzung „.*“ wird nicht genutzt.

richtig

```
import java.util.ArrayList;
import java.util.Iterator;
```

falsch

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

- Auf static import zur vereinfachten Nutzung von Klassenmethoden wird verzichtet.

richtig

```
import
  org.junit.jupiter.api.Assertions;
```

und im Programm

```
Assert.assertTrue(tmp>42);
```

falsch

```
static import
  org.junit.jupiter.api.Assertions;
```

und im Programm

```
assertTrue(tmp>42);
```

- Sollen Methoden einer Klasse K getestet werden, so hat die Testklasse immer den Namen KTest.

- Soll eine Methode mit Namen methode getestet werden, so hat die Testmethode den Namen testMethode, soll die Methode in verschiedenen Tests geprüft werden, wird der Name der Testmethode durch eine Zahl (testMethode1) oder besser das erwartete Verhalten (testMethodeAusnahmeBeiNull) ergänzt.

Glossar

Die Begriffserklärungen werden aus Sicht der Veranstaltung „Grundlagen der Programmierung“ geschrieben und sind hierfür ausreichend; an wenigen Stellen wäre die vollständige Erklärung etwas komplexer (z. B. bei Interfaces), da aber nicht alle eventuellen Möglichkeiten beim ersten Kontakt mit objektorientierter Programmierung eine Rolle spielen, werden diese leicht vereinfachten Erklärungen gegeben. Das Glossar ist trotzdem für andere Veranstaltungen nutzbar.

Begriff	Bedeutung / Anmerkung
abstrakte Klasse	eine *Klasse* in der mindestens eine *ObjektMethode* durch das *Schlüsselwort* `abstract` gekennzeichnet und deshalb nicht ausimplementiert ist; vor dem Klassennamen steht ebenfalls das Schlüsselwort `abstract`; von abstrakten Klassen können keine *Objekte* direkt erzeugt werden, dies geschieht durch Klassen, die diese be*erben* und alle abstrakten Methoden realisieren, generell müssen abstrakte Methoden in der erbenden Klasse nicht realisiert werden; wenn dies nicht geschieht, muss die Klasse auch abstrakt sein; da *Konstruktor*enaufrufe mit super() bei Vererbung eine Rolle spielen, können abstrakte Klassen sinnvolle Konstruktoren haben
Aktivitätsdiagramm	Teil der Modellierungssprache UML zur Beschreibung von Abläufen mit Visualisierungsmöglichkeiten für einfache Schritte (Aktionen), Auswahlalternativen und Schleifen; in der vollständigen Variante bestehen weitere Modellierungsmöglichkeiten
Algorithmus	ist ein Verfahren (genauer die Beschreibung eines Verfahrens) zur Lösung einer gegebenen Aufgabe, diese Beschreibung ist endlich
Alternative	Programmstück, mit dem in Abhängigkeit von einer *Booleschen Bedingung* ein folgendes Programmstück ausgeführt werden kann (`if` bzw. `if else` bzw. `switch case`)
Annotation	wird dazu genutzt, um weitere *Eigenschaft*en von *Klasse*n, *Variable*n, *Parameter*n und *Methode*n festzulegen, die Annotation `@Override` z. B. fordert, das die nachfolgende Methode eine Methode der *beerbten Klasse* überschreibt
Array	Spezialfall einer *Sammlung* von *Objekt*en eines bestimmten *Typ*s, für den bereits bei der ersten Nutzung bekannt ist, wieviele Objekte maximal verwaltet werden sollen; hat in Java ein besondere Schreibweise, z. B. `String[] monatsnamen = new String[12];`
Attribut	s. Objektvariable
Aufzählung	eine Liste von konkreten Werten, die einen eigenen *Typ*en definieren, wie es z. B. für Ampelfarben rot, rotgelb, gelb und gruen der Fall ist; in Java werden Aufzählungen als enumeration mit dem *Schlüsselwort* `enum` realisiert, die normale *Klasse*n sind und so auch *Methode*n enthalten können

Begriff	Bedeutung / Anmerkung
Ausdruck	ist anschaulich etwas, was ausgerechnet werden kann, wie `7+4` und `"Hal"+"lo"` und hat einen Ergebnis*typ*en; Ausdrücke stehen häufig auf der rechten Seite von *Zuweisung*en und werden als *Boolesche Bedingung*en in *Alternative*n und *Schleife*n genutzt. Oft wird der Ergebnistyp dem Begriff vorangestellt, z. B. int-Ausdruck bei `7+4` oder auch `"Hallo".length()`
Ausnahme	spezielles *Objekt*, das bei der Ausnahmebehandlung genutzt wird; Ausnahmen (Exceptions) werden im Programm geworfen, wenn eine außergewöhnliche Situation auftritt, die vom erwarten Ablauf abweicht (z. B. soll eine Datei geöffnet werden, die unerwartet nicht gelesen werden darf); Ausnahmen können in try-catch-finally Programm*blöcken* behandelt werden, Ausnahmen werden bei Nichtbehandlung an aufrufende *Methode*n weitergeleitet; entwickelnde Personen können selbst Ausnahmen als *Klasse*n programmieren; es werden Ausnahmen unterschieden, die mit try-catch behandelt werden müssen und welche, für die dies optional (und teilweise sinnlos) ist
Autoboxing	spezieller Ansatz in Java, mit dem es unerheblich sein soll, ob mit echten Integer-Objekten oder mit primitiven int-Werten gearbeitet wird, da, wenn möglich, eine automatische Umwandlung passiert; löst aber nicht alle Probleme
beerbte Klasse	*erb*t eine *Klasse* B von einer Klasse A, in Java `public class B extends A`, heißt A die beerbte Klasse

Block	beschreibt formal den Programmcode innerhalb eines Paares von öffnender und schließender geschweifter Klammer, die immer paarweise und ineinander geschachtelt auftreten.
	Beispiel:
	`public int eineObjektmethode(int jahr){ /* Blockbeginn 1 */`
	`  if (jahr>2000) { /* Blockbeginn 2 */`
	`  return 0;`
	`  } /* Blockende 2 */ else { /* Blockbeginn 3 */`
	`  return 1:`
	`  } /* Blockende 3 */`
	`} /* Blockende 1 */`

Boolesche Bedingung	können nach `true` (wahr) oder `false` (falsch) ausgewertet werden, sie können Boolesche Variablen sein oder Ausdrücke, die *Variable*n beinhalten (`x>3`); Boolesche Bedingungen können durch logische Operatoren (Junktoren: nicht, und, oder) zu komplexeren Booleschen Bedingungen kombiniert werden; nach George Boole (* 2. November 1815, † 8. Dezember 1864) alternativ: Boolescher Ausdruck

Begriff	Bedeutung / Anmerkung
casten	von der entwickelnden Person gesteuerte Umwandlung eines *Objekt*s eines *Typ*s A in ein Objekt eines anderen Typs B; dies ist nur möglich, wenn B von A *erbt* und das *referenz*ierte Objekt ursprünglich vom Typ B oder einem von B erbenden Typ angelegt wurde; da die Überprüfung der Möglichkeit erst zur Laufzeit geschieht, liegt die Verantwortung für die Korrektheit bei der entwickelnden Person; in klassischen Programmen soll auf casten verzichtet werden, da dies oft ein Indikator für die unsaubere Nutzung von Vererbung ist
Collection	s. Sammlung
Compiler	Teil des Übersetzungsverfahrens vom Programmcode zum ausführbaren Programm, der aus dem ge*parsten*, also syntaktisch korrekten Programm neuen Programmcode erzeugt, der entweder direkt vom Computer ausgeführt werden kann (ist so bei C, C++) oder in eine Zwischensprache überführt, die einfach von einer *virtuellen Maschine* ausgeführt werden kann (ist so in Java, C#)
Debugger	Werkzeug der Entwicklung, mit dem man die Programmausführung unterbrechen kann, um sich dann genau die Werte aller bis dahin definierten *Variable*n anzusehen und das Programm dann schrittweise weiter ausführen zu können; einige Debugger erlauben die Veränderung von *referenz*ierten *Objekt*en
Design by Contract	kann als „Entwurf nach Regeln eines Vertrags“ übersetzt werden, dabei wird in der Entwicklung frühzeitig festgelegt, was eine *Klasse* leisten soll, dazu müssen aber nur die die Namen der *Methode*n und die zugehörigen *Parameterliste*n, sowie das gewünschte Verhalten festgelegt werden, was z. B. durch *Interface*s oder *abstrakte Klassen* möglich ist; Nutzer der Klassen können sich dann darauf verlassen, dass es sinnvolle Realisierungen dieser „vertraglich vereinbarten“ Schnittstelle gibt
dynamische Polymorphie	erbt eine *Klasse* K2 von einer anderen Klasse K1, kann sie *Methode*n der Klasse K1 *überschreiben*; *Variable*n vom *Typ* K1 können dann *Objekt*e der Klassen K1 und K2 *referenz*ieren, wird dann die überschriebene Methode ausgeführt, wird zur Laufzeit (deshalb dynamisch) bestimmt, welche der Methoden es ist, die in der am weitesten abgeleiteten Klasse ausgeführt wird
Eigenschaft	hat einen Namen, einen Typen und kann verschiedene Werte annehmen
elementarer Typ	s. primitiver Datentyp
Entwicklungsumgebung	Software, die die Erstellung einer Software unterstützt, dies umfasst einen Editor und typischerweise die Möglichkeit, einen *Compiler* zu nutzen und die Fehlermeldungen des *Parser* systematisch abzuarbeiten; weiterhin können in einer integrierten Entwicklungsumgebung weitere Arbeitsschritte, wie die Nutzung eines *Debugger*s, die Modellierung und die Verwaltung verschiedener Versionen des Programmcodes mit unterstützt werden
Enumeration	s. Aufzählung
erben	s. vererben

Begriff	Bedeutung / Anmerkung
erbende Klasse	*erb*t eine *Klasse* B von einer Klasse A, in Java `public class B extends A`, heißt B die erbende Klasse
Exemplarvariable	s. Objektvariable
Exception	s. Ausnahme
Feld	s. Array
Genauigkeit	beinhaltet, dass beim Rechnen mit Kommazahlen nicht immer genau gerechnet werden kann; gerade bei der Verknüpfung von sehr großen und sehr kleinen Zahlen können größere Abweichungen auftreten
Gleichheit	bedeutet die inhaltliche Überprüfung, ob zwei *Objekt*e als gleich behandelt werden sollen, in Java wird dies über die möglichst für jede *Klasse* zu *überschreiben*de Methode `public boolean equals(Object o)` geregelt; unterscheidet sich üblicherweise von der *Identität*
`hashCode`	Methode, die Java bei einigen Klassen deren Name meist mit „Hash“ beginnt, zur schnelleren Prüfung auf Gleichheit genutzt wird; nur wenn für beide Objekte der HashCode gleich ist, wird die Methode equals zur Prüfung der Gleichheit genutzt; generell sinnvoll die Methode `public int hashCode()` zu überschreiben
Identität	bedeutet die Überprüfung, ob zwei *Variable*n das gleiche *Objekt referenz*ieren, dies wird in Java mit `==` geprüft und ist von der Objekt-*Gleichheit* zu unterscheiden
immutable object	s. nicht veränderbare Objekte
inkrementelle Entwicklung	Entwicklung eines komplexen Systems durch die Realisierung mehrerer Teilfunktionalitäten hintereinander, die Software wird dabei schrittweise um neue Funktionalität ergänzt, z. B. gibt es erst die Möglichkeit, Studierende zu speichern und dann die Möglichkeit, Studierenden Noten zuzuordnen; verwandt mit dem Begriff *iterative Entwicklung*
Instanz	s. Objekt
Instanzvariable	s. Objektvariable
Interface	eine völlig *abstrakte Klasse*, in der keine *Methode* eine Realisierung hat, das *Schlüsselwort* `abstract` kann deshalb weggelassen werden; diese Spezialform einer abstrakten Klasse erlaubt in Java eine *Mehrfachvererbung* und wird oft als Schablone bei einer Entwicklung mit dem Absatz „*Design by Contract*“ bezeichnet, die dann durch Klassen, die in einer bestimmten Form genutzt werden sollen, realisiert werden; in Java kann eine Klasse durch das Schlüsselwort `implements` angeben, welche Interfaces realisiert werden, Interfaces werden mit dem Schlüsselwort `interface` gekennzeichnet; Java erlaubt es seit der Version 8, dass Methoden in Interfaces mit dem *Schlüsselwort* `default` gekennzeichnete Implementierungen besitzen, die dann in der das Interface realisierenden Klasse überschrieben werden können, aber nicht müssen

Begriff	Bedeutung / Anmerkung
iterative Entwicklung	Entwicklung eines Programms oder einer *Methode* durch schrittweise Verfeinerung; es werden z. B. erst das typische Verhalten implementiert und dann schrittweise *Alternative*n und *Ausnahme*n ergänzt
Iterator	eigene *Klasse*, deren *Objekt*e zum effizienten Durchlaufen von *Sammlung*en genutzt wird; anschaulich handelt es sich um eine Referenz, die immer auf genau ein Element der Sammlung zeigt; mit der Methode `next()` wird dieses Element ausgegeben und gleichzeitig auf ein nachfolgendes Element der Sammlung gesetzt, mit `hasNext()` kann gefragt werden, ob beim Aufruf von `next()` ein Objekt zurückgegeben wird oder ob das Ende der Sammlung erreicht wurde
Klasse	Struktur zur Beschreibung von *Objekt*en, in Java am *Schlüsselwort* `class` erkennbar; es werden die *Eigenschaft*en von Objekten durch *Objektvariable*n definiert, die jedes Objekt der Klasse haben soll; weiterhin werden alle *Objektmethode*n notiert, die auf Objekten ausgeführt werden können; enthält mindestens einen *Konstruktor*, mit dem Objekte erzeugt werden können; bis auf Konstruktoren und auch enthaltene *Klassenmethoden* kann der Programmcode der Klasse nicht ausgeführt werden; zur Ausführung von Objektmethoden werden Objekte der Klasse benötigt, die über einen Aufruf eines Konstruktors mit `new` erzeugt werden können
Klassenbibliothek	Zusammenfassung von mehreren *Klasse*n, meist logisch gruppiert in *Paket*en, die den entwickelnden Personen bei der Programmierung zur Verfügung stehen; Java bietet mit der Java-Klassenbibliothek eine sehr mächtige Grundlage zur Programmierung
Klassenmethode	ist *Methode* der *Klasse* und nicht der einzelnen *Objekt*e, wird typischerweise zum Lesen und Verändern von *Klassenvariable*n genutzt; die Methoden sind auch ohne Objekte der Klasse nutzbar und werden in der Form K.m() für eine Klasse K mit einer Klassenmethode m() aufgerufen; innerhalb von Klassenmethoden kann nicht auf *Objektvariable*n zugegriffen werden; in Java am Schlüsselwort `static` erkennbar
Klassenvariable	ist die *Eigenschaft* der *Klasse* und nicht eines einzelnen *Objekt*es, ist mit dem *Schlüsselwort* `static` markiert; kann z. B. zum Erzeugen eindeutiger Nummern genutzt werden, wenn die Klassenvariable bei jeder Objekterstellung hochgezählt wird; sie können auch zur Deklaration von *Konstante*n genutzt werden; sind dürfen nicht zum Austausch von Werten zwischen Objekten einer Klasse genutzt werden
Kommentar	wird grundsätzlich vor dem zu beschreibenden Element (*Variable*, *Methode*, *Klasse*) angegeben; dabei wird beschrieben, wozu das Element dient, bei Methoden werden alle *Parameter* und bei einem Rückgabetyp ungleich void das Ergebnis genau beschrieben; der übliche Kommentar ist mehrzeilig und steht zwischen `/*` und `*/`; kurze Kommentare bis zum Zeilenende werden durch `//` eingeleitet; ein professionelles Programm existiert nur mit professionellen Kommentaren; aus den Kommentaren kann meist ein wesentlicher Teil der Programmdokumentation generiert werden (JavaDoc, Doxygen)

Begriff	Bedeutung / Anmerkung
Konstante	Generell können *Variable*n beliebig veränderbare *Objekte referenz*ieren, möchte man aber, dass ein Wert unveränderbar ist, wie z. B. der Firmenname, wird mit dem *Schlüsselwort* `final` festgelegt, dass diese Variable nicht verändert werden kann, also konstant bleibt und nicht auf der linken Seite von *Zuweisung*en auftreten kann
Konstruktor	Spezielle Art einer *Methode*, mit der es möglich ist, ein *Objekt* einer *Klasse* zu erzeugen; nur für so erzeugte Objekte können dann Objektmethoden ausgeführt werden, bekommen Klassen als statische Schablonen ihren eigentlichen Sinn
Kopie	Da Java ausschließlich mit *Referenz*en arbeitet, werden auch bei *Zuweisung*en von Sammlungen wie `ArrayList<Integer> s1 = s2` nur neue Referenzen auf die gleichen *Objekt*e angelegt; möchte man echte Kopien von Objekten herstellen, muss man dies explizit realisieren und dafür sorgen, dass auch von allen *Objektvariablen* Kopien angelegt werden; Java unterstützt dies teilweise mit dem `clone()`-Befehl, der aber ohne *Überschreiben* nur die direkt referenzierten Objekte, aber wiederum nicht deren referenzierte Objekte kopiert; alternativ kann man über einen Kopier*konstruktor* nachdenken, der ein Objekt übergeben bekommt und eine echte Kopie des Objekts herstellt
Lebensspanne	bezeichnet den Teil eines Programms, in dem eine *Variable* genutzt werden kann, dies ist typischerweise der *Block*, in dem die Variable definiert ist
lokale Variable	*Variable*, die innerhalb einer *Methode* definiert wird und genutzt werden kann; genauer werden lokale Variablen innerhalb eines *Blockes*, den direkt umgebenden geschweiften Klammern, deklariert und können genau in diesem Block genutzt werden; nach Verlassen des Blockes „stirbt“ die lokale Variable und ist nicht mehr zugreifbar; ein *referenz*iertes Objekt wird gelöscht, insofern keine andere Variable mehr dieses Objekt referenziert
Mehrfachvererbung	schafft die Möglichkeit, dass eine *Klasse* gleichzeitig von mehreren anderen Klassen *erben* kann; da dies durch mehrfaches Erben von der gleichen Klasse über verschiedene Wege recht kompliziert werden kann, ist eine direkte Mehrfachvererbung in Java nicht erlaubt; sie wird aber durch die erlaubte Realisierung von mehreren *Interface*s unterstützt
Methode	zusammenfassender Begriff für *Objektmethode*n und *Klassenmethode*n

nicht veränderbare Objekte	spezielle *Objekt*e, die über *Referenz*en nicht verändert werden können, in Java gehören Objekte der Zahl-*Klasse*n Integer und Double sowie String dazu
	Beispiel: anders als bei veränderbaren Objekten, was der typische Fall ist, können über zwei *Variable*n, die das gleiche Objekt referenzieren, nicht die Inhalte des Objekts verändert werden.
	`String s1 =“Hai“;`
	`String s2 = s1;`
	`s1 = s1.replaceAll(“i“,“llo“);`
	Danach hat `s1` eine neue Referenz auf ein Objekt „Hallo“, `s2` referenziert weiter ein Objekt „Hai“.

Begriff	Bedeutung / Anmerkung
null	sogenannte Nullreferenz, die kein Objekt referenziert; die beste Veranschaulichung ist der Wert „undefiniert“; nur Variabler deren Typ eine Klasse ist, können den Wert `null` haben
Oberklasse	s. beerbte Klasse
`Object`	spezielle *Klasse* in Java, von der ohne weitere Angaben alle anderen Klassen *erben* und so automatisch eine Realisierung der *Methode*n `toString()` und `equals(.)` haben, die dann sinnvoll überschrieben werden sollten
Objekt	eindeutiges Individuum, dass durch seine *Eigenschaft*en beschrieben wird; Objekte entstehen durch den Aufruf eines *Konstruktor*s einer *Klasse* Beispiel: Studentin mit Eigenschaften Name Anna und Matrikelnummer 424242 alternativer Begriff: Instanz
Objektmethode	*Methode*n, die in einer *Klasse* definiert werden und die nur für *Objekte* dieser Klasse (oder Objekten von Klassen, die diese Klasse be*erbt* haben) ausgeführt werden können; nur innerhalb von Objektmethoden kann auf *Objektvariablen* und mit *this* auf das Objekt selbst zugegriffen werden; unterscheiden sich von *Klassenmethode*n, in deren Deklaration das Schlüsselwort `static` steht
Objektvariable	definiert formal in einer Klasse eine Eigenschaft der dort beschriebenen Objekte; hat einen Namen und einen Typen alternative Begriffe in der Literatur: Exemplarvariable, Instanzvariable, Attribut
Paket	meist werden eine oder mehrere *Klasse*n logisch in einem Paket zusammengefasst; anschaulich kann man sich hier Dateiordner vorstellen, die dann die Klassen enthalten; weiterhin können Pakete auch andere Pakete, auch zusammen mit Klassen, enthalten; in Java am Anfang der Datei mit dem Schlüsselwort `package` angegeben
Parameter	spezielle Art von *Variable*, die in der Definition von *Konstruktor*en und *Methode*n steht, wird deshalb oft auch Übergabeparameter genannt; jeder Parameter hat einen *Typ* und einen Namen und kann im folgenden Programm*block* genutzt werden; beim Aufruf von Konstruktoren und Methoden werden beim Aufruf bekannte Variablen genutzt, durch den Aufruf referenzieren dann Parameter deren Typ eine *Klasse* ist die gleichen Objekte wie diese Variablen; bei Parametern mit *primitiven Typ*en werden die Werte der übergebenen Variablen kopiert
Parameterliste	kommaseparierte, mit einer öffnenden runden Klammer beginnende und einer schließenden runden Klammer endende Liste von *Parameter*n
Parser	Teil des Übersetzungsverfahrens vom Programmcode zum ausführbaren Programm, der die Einhaltung der Syntaxregeln, d. h. wie ein korrektes Programm vom Text her aussehen kann, überprüft

Begriff	Bedeutung / Anmerkung
primitiver Datentyp	spezielle Art eines *Typ*s, der keine *Klasse* ist und nur einfache Werte und keine *Objekt*e aufnehmen kann und eigentlich in rein objektorientierten Sprachen (z. B. Smalltalk, Ruby) nicht vorkommt, aber Bestandteil von Java, C#, C++ und C ist; klassische Beispiele sind `int`, `long`, `double`, `float`, `char`, `boolean`
Referenz	bezieht sich auf ein Objekt das sich im Speicher befindet, wird durch eine Variable über den Variablennamen nutzbar; es kann mehrere Variablen geben, die das gleiche Objekt referenzieren
Reihung	s. Array
Sammlung	ist der Oberbegriff für verschiedene *Typ*en, die Mengen von *Objekt*en verwalten können, ein Beispiel ist `ArrayList<Integer>` zur Verwaltung einer Menge von Integer-Objekten; neben Listen können auch Mengen (Set) oder Zuordnungen (Map) oder Multimengen genutzt werden; Sammlungen unterscheiden sich danach wie oft ein Objekt vorkommen darf und ob die Reihenfolge relevant ist; ein *Array* ist ein Spezialfall einer Sammlung, bei dem bereits bei der ersten Nutzung bekannt ist, wie viele Objekte maximal aufgenommen werden dürfen
Schleife	dient zur wiederholten Ausführungen von Anweisungen des nachfolgenden *Block*s, die Anzahl der Wiederholungen hängt von einer *Booleschen Bedingung* ab, es gibt die Varianten der `while`-, `do-while`- und `for`-Schleifen
Schlüsselwort	spezielle Wörter, die in der Programmiersprache für Befehle genutzt werden und deshalb z. B. nicht als *Variable*nnamen zur Verfügung stehen, einige Beispiel in Java sind `public`, `class`, `import`, `extends`, `implements`, `throws`, `throw`, `if`, `else`, `while`, `default`, `abstract`
Serialisierung	beschreibt die Möglichkeit, *Objekt*e einfach in Dateien zu schreiben und ebenfalls schnell wieder herauszulesen; passiert typischerweise im Binärformat, was beim Ändern der *Klasse*n dazu führen kann, dass Objekte nicht mehr gelesen werden können; Java bietet mit den Klassen XMLEncoder und XMLDecoder eine komfortable Möglichkeit, Objekte im XML-Format zu speichern und wieder zu laden
Sichtbarkeit	kann für *Objektvariable*n, *Objektmethode*n, *Klassenvariable*n, *Klassenmethode*n und *Klasse*n angegeben werden und definieren, welcher Zugriff erlaubt ist; `public` steht dabei für Zugriff durch alle, `protected` für den Zugriff nur durch erbende Klassen und `private` für den Zugriff nur innerhalb der Klasse; lässt man in Java die Sichtbarkeit weg, kann ein Zugriff aus dem gleichen Paket erfolgen
Software-Architektur	beschreibt den Aufbau der Software und gewährleistet mit die Wart- und Erweiterbarkeit, da bestimmte Funktionalität in bestimmten *Paket*en (z. B. Oberfläche) organisiert wird und Regeln aufgestellt werden, welche *Klasse*n *Methode*n welcher anderen Klassen in welchen Paketen nutzen dürfen

Begriff	Bedeutung / Anmerkung
Sonderzeichen	Zeichen, die nicht unmittelbar durch die Tastatur, genauer amerikanische Tastatur (ASCII) eingegeben werden können und deshalb eine besondere Darstellung haben; dies können Steuerzeichen zum Zeilenumbruch `\n`, aber auch Umlaute `\u00d6` für Ö sein; Java nutzt den Unicode-Zeichensatz

statische Polymorphie	eine *Klasse* kann mehrere *Konstruktor*en und mehrere *Methode*n mit gleichen Namen haben, solange sich die *Parameterliste*n bzgl. der *Typ*en, zumindest in der Reihenfolge unterscheiden; dadurch kann beim *kompilieren* anhand der Parameter bei der Nutzung entschieden werden, welcher Konstruktor, bzw. welche Methode aufgerufen wird, erlaubt sind z. B. die folgenden Methoden
	`public void machen (String name, int alter) { /* ... */ }`
	`public void machen (int alter, String name) { /* ... */ }`

Stream	übersetzbar als Datenfluss; bezeichnet, dass Informationen Stück für Stück nacheinander bearbeitet, also gelesen oder geschrieben werden dürfen, Beispiele sind das Lesen von Tastatureingaben, das Lesen von Dateien und das Schreiben in Dateien; der Begriff hat ab Java 8 eine besondere Bedeutung, da auch *Sammlung*en als Streams der enthaltenen *Objekt*e verarbeitet werden können
Testfall	beschreibt einen konkreten Test, dazu wird angegeben, welche Situation vor dem Test hergestellt werden muss, wie der Test durchgeführt werden muss (typischerweise *Methodenaufruf*) und welche Ergebnisse erwartet werden; in JUnit wird hierzu die *Annotation* `@Test` genutzt
Testfixture	wird für mehrere *Testfälle* die gleiche Ausgangssituation benötigt, spricht man von einer Testfixture; in JUnit wird hierzu die *Annotation* `@BeforeEach` genutzt
this	*Referenz* auf das *Objekt* selbst, in dessen *Objektmethode*n es genutzt wird; dabei ist `this` das Objekt selbst und this.objektvariable eine Referenz auf die *Objektvariable* des betrachten Objekts; es folgt, dass die Nutzung von this nur innerhalb von Objektmethoden Sinn macht
Typ	jede *Variable* in Java (C, C#, C++) hat einen konkreten Typen, diese Variablen können nur Werte dieses Typs oder von *Oberklasse*n des Typs annehmen; in der reinen *Objekt*orientierung entsprechen Typen den *Klasse*n, in Sprachen wie Java, C++ und C# gibt es auch *primitive Typ*en wie `int` und `char`, die keine Objekte sind; Typ ist damit die Zusammenfassung der Begriffe Klasse und primitiver Typ
überschreiben	*erbt* eine *Klasse* K2 von einer Klasse K1, können für *Objekt*e der Klasse K2 alle *Methode*n mit den *Sichtbarkeiten* `public` und `protected` von Objekten von K1 nutzen; weiterhin besteht die Möglichkeit, eine neue Implementierung der Methode zu schreiben, wobei der gleiche Methodenname und eine *Parameterliste* mit genau den gleichen *Typ*en in gleicher Reihenfolge genutzt werden muss; diese Neuimplementierung wird auch Überschreiben der Methode genannt

Notiz

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

Begriff	Bedeutung / Anmerkung
Unit Test	beschreibt das Testen einer einfachen Einheit, meist einer Methode oder das einfache Zusammenspiel von Methoden, dabei wird die zu testende Methode unter vorher festgelegten Randbedingungen ausgeführt und geprüft, ob das Ergebnis die gewünschten Eigenschaften hat; in Java wird zur Ausführung von *Testfäll*en JUnit dazu genutzt, womit allerdings noch weitere Testarten durchgeführt werden können
Variable	Name, der einen Wert eines bestimmten *Typ*s annehmen kann, genauer handelt es sich um eine *Referenz* auf ein *Objekt*; gibt es in den verschiedenen Varianten: *Objektvariable*, *lokale Variable*, *Parameter*, *Klassenvariable*, *Konstante*
vererben / Vererbung	erweitert eine *Klasse* B eine Klasse A, in Java `public class B extends A`, können in B alle *Variable*n und *Methode*n mit den *Sichtbarkeit*en `public` und `protected` von A direkt genutzt werden, in B können auch Methoden von A *überschrieben* werden, man sagt „B erbt von A“, auch die *Oberklasse* A vererbt ihr Verhalten an B
vererbende Klasse	s. beerbte Klasse
virtuelle Maschine	Software zur Ausführung von Programmen, die in einem bestimmten maschinennahen Format, in Java in Byte Code, vorliegen, der durch die virtuelle Maschine ausgeführt wird; unterscheidet sich vom Ansatz, dass Programme direkt in ausführbare Programme *compiliert* werden; der gleiche Java Byte Code läuft auf allen großen Betriebssystemen, da nur betriebssystem-individuelle virtuelle Maschinen benötigt werden
Zuweisung	hat immer die Form, dass auf der linken Seite eine *Variable*, gefolgt von einem Gleichheitszeichen und einem Ausdruck auf der rechten Seite besteht, dabei kann der Ausdruck auch eine Variable (`this.x = this.y`) oder ein zu berechnender Ausdruck (`this.x = p.getX()+42`) sein