*Struktur eines Java-Programms*

Im Zusammenhang mit der objektorientierten Programmierung haben Begriffe wie Klasse, Objekt, Attribut, Methode, Vererbung und Interface eine besondere Bedeutung.

*Klasse:*

* Klassen definieren neue Typen, die als Programmierer komplett an die eigenen Bedürfnisse zuschneiden können.
* sind wesentlich leistungsfähiger als primitive Typen
* können eine vielzahl von Werten speichern, die ihren Zustand als Eigenschaften beschreiben
* können botschaften reagieren und selbst aktiv werden

Beispiel:

class Brunch {

int zaehler;

int nenner;

}

*Package:*

* Packages dienen dazu, mehrere logisch zusammengehörige Klassen zusammenzufassen und damit die Verwaltung größerer Programme zu vereinfachen.

Beispiel:

package uebung05;

class Bruch {

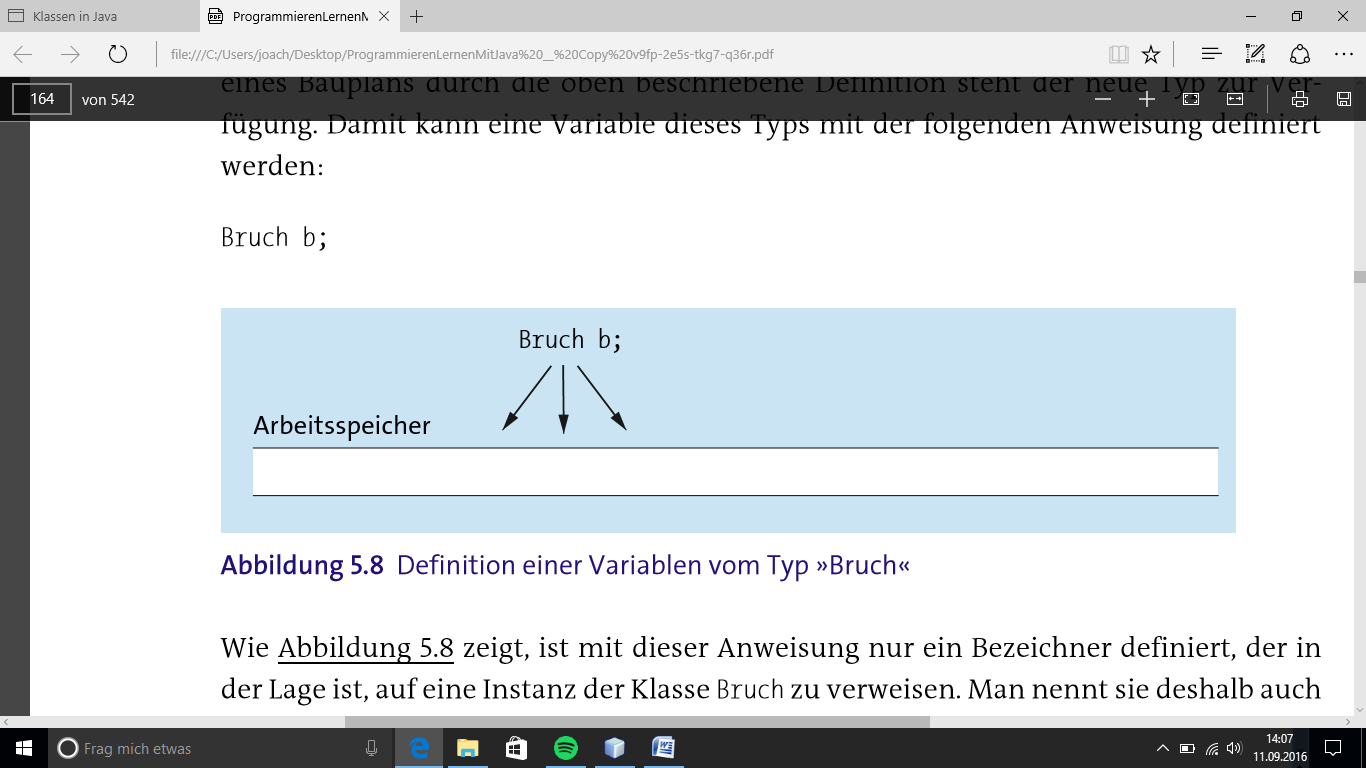
int zaehler;

int nenner;

}

*Objekt:*

* Ein Objekt ist ein Exemplar, das nach dem Bauplan einer Klassendefinition erstellt wurde. Die Klasse stellt den Bauplan dar. Das Objekt ist eine Variable, die nach diesem Plan aufgebaut ist.
* Es können beliebig viele Objekte (Instanzen) erzeugt werden, die Instanzen weisen aber in der Handhabung deutliche Unterschiede auf.
* im Arbeitsspeicher ist unmittelbar auch Speicherplatz verfügbar, auf den über den Bezeichner zugegriffen weden kann



* nur ein Bezeichner ist definiert
  + if (b != null) …
* ist in der Lage, auf eine Instanz der Klasse Bruch zu verweisen
* noch kein Platz im Hauptspeicher für die Atribute zaehler und nenner reserviert
* es existiert noch keine Instanz
* Variable b hat den vordefinierten Wert null
* nur wenn die Bedingung (b ! = null)
* Wir der Wert true zurückgeliefert existiert bereits eine Instanz der Klasse Bruch
* neue instanz der Klasse Bruch erzeugen mit den Operator new
  + new Bruch();
* Verbindung zum Bezeichner
  + Bruch = new Bruch();

*Attribut:*

Die Attribute eines Objektes werden über den Objektnamen angesprochen. Auf den Objektnamen folgt, durch einen Punkt getrennt, der Name des Attributs. Diese Zugriffsmöglichkeit kann und soll sogar vom Programmierer unterbunden werden. Sie wird hier nur der Vollständigkeit halber beschrieben.

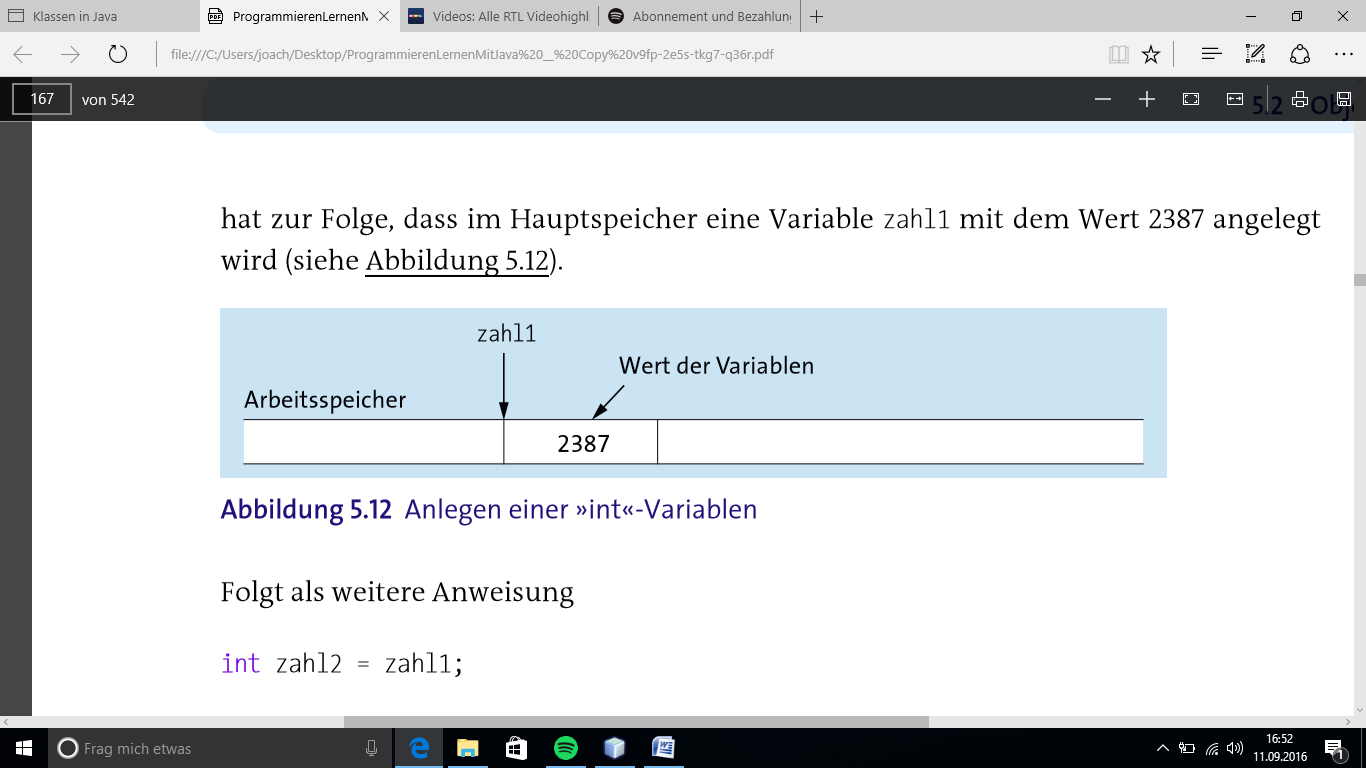
* Inkrementierung des Zählers 🡪 b.zehler++;
* Prüfen, ob der Nenner ungleich null ist 🡪 if (b.nenner != 0)

An dem Punkt zwischen Objektbezeichner und Datenelementbezeichner kann man erkennen, dass hier mit dem Attribut eines Objekts und nicht mit einer lokalen Variablen gearbeitet wird.

*Wertzuweisungen bei Objekten:*

* Objekte sind Referenztypen.
* Eine Wertzuweisung hat andere Folgen als bei den primitiven Typen

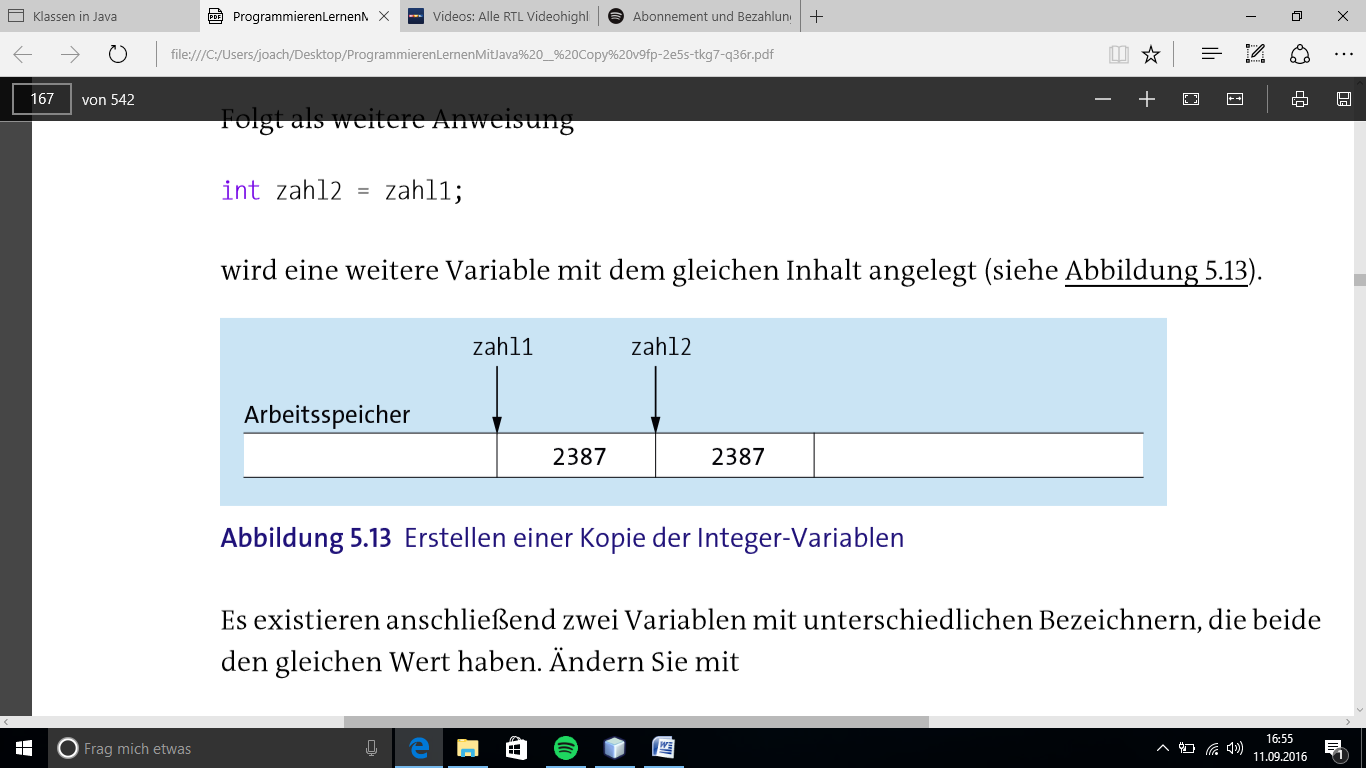
int zahl1 = 2387 🡪 im Hauptspeicher eine Variable zahl 1 mit dem Wert 2387



Weitere Anweisung

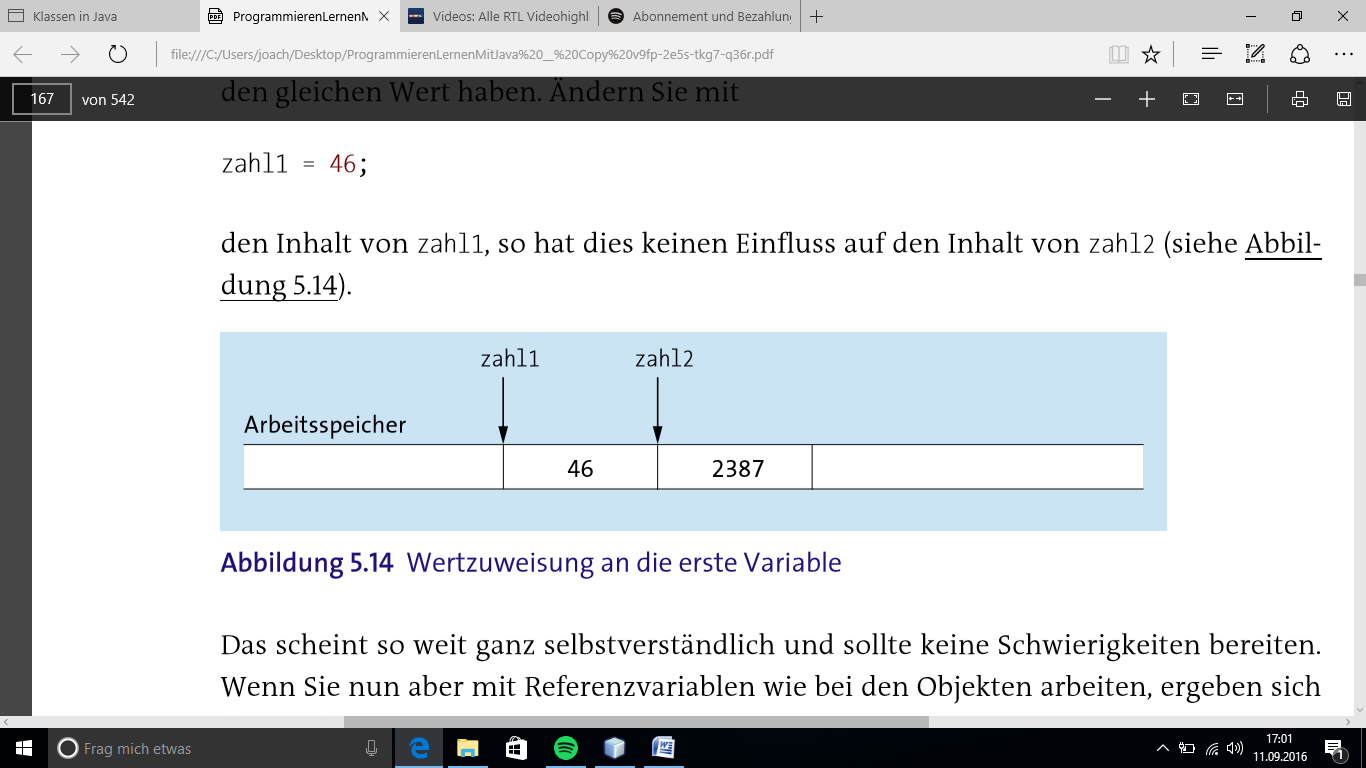
int zahl2 = zahl1;

weitere Variable mit dem gleichen Inhalt:



Wert ändern mit:

zahl1 = 46



Aliasing

b.zaehler = 5;

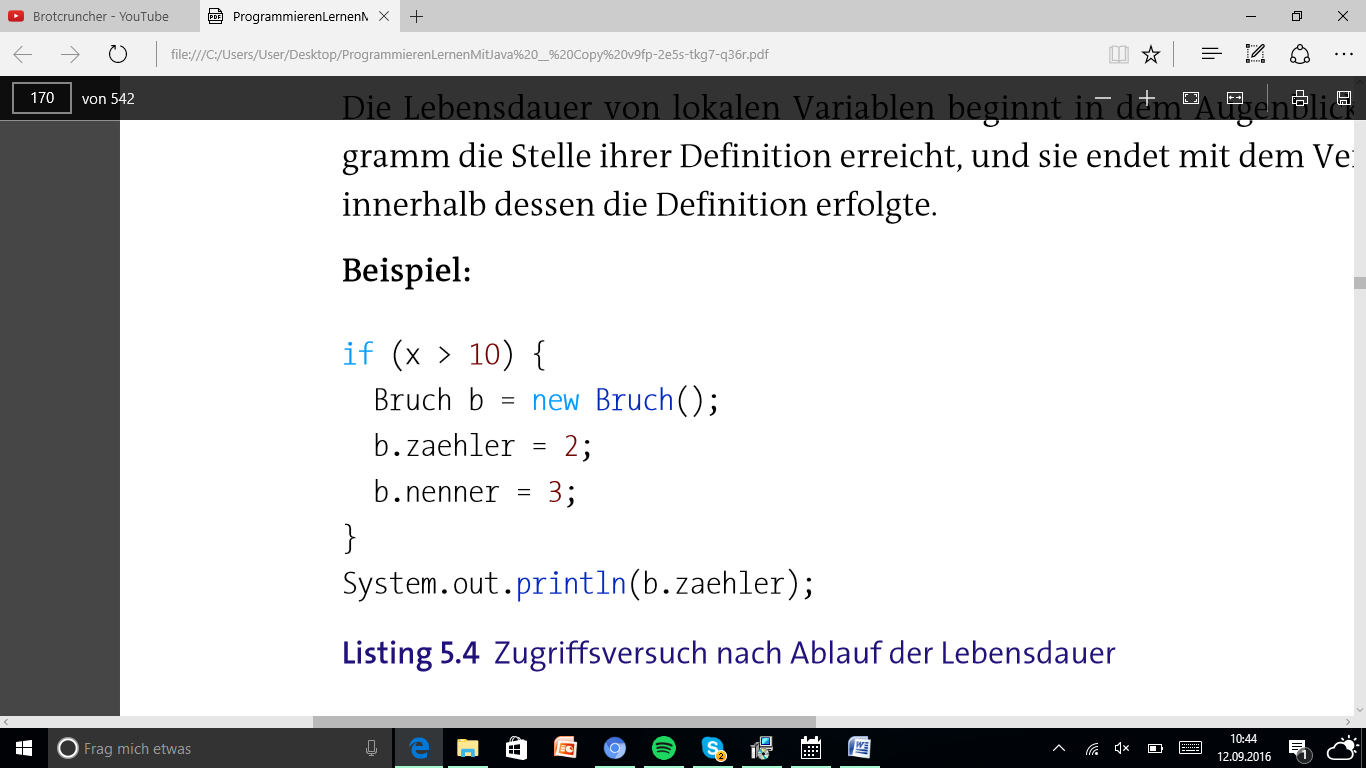
b.nenner = 8;

Vergleiche von Objekte mit Aliasing:

if(a == b) …

*Gültigkeitsbereich und Lebensdauer:*

* Unterschiede zwischen lokalen Variablen und Datenelementen eines Objekts
* gelten nur innerhalb des Blocks, in dem sie Definiert wurden
* Attributes eines Objektes haben unabhängig innerhalb der gesamten Klasse Gültigkeit
* Werden beim definieren gültig
* enden beim Verlassen des Blocks



* Bruch b wird erzeugt, wenn x größer 10
* Lebensdauer endet nach dem schließen der Klammer
* danach ist kein Zugriff mehr auf die Variable
* System.out.println(b.zaehler); wird einen Fehler verursachen

Lebensdauer endet automatisch, sobald es im Programm keine Referenz mehr auf das Objekt gibt.

Wenn keine Referenz mehr besteht und der verfügbare Speicherplatz zuneige geht.

*Methoden:*

* beschreiben das Verhalten einer Klasse

Aufbau

Kopf:

* Überschrift
* Rückgabetyp
* Bezeichner der Methode
* in Runden klammern: Datentypen und Bezeichner von Übergabeparametern

Rumpf:

* legt fest welche Vorgänge mit dem Aufruf der Methode ablaufen sollen
* Java- Anweisungen

Beispiel:

void ausgeben() { 🡪 Kopf der Methode

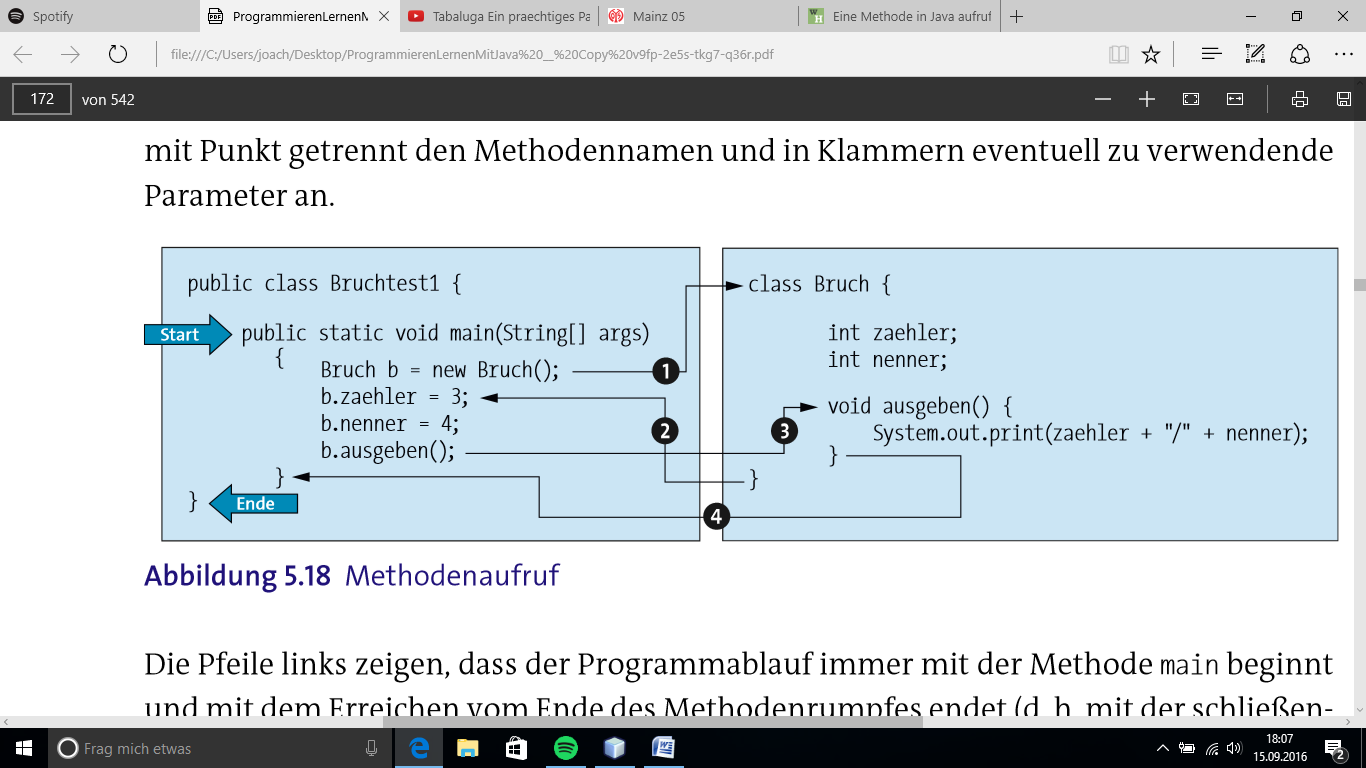
System.out.print(zaehler + “/“ + nenner); 🡪Rumpf der Methode

Methoden können Daten als Ergebnis zurückliefern

*Aufruf von Methoden*

Instanzmethode 🡪 ausgeben()

* Punkt trennt den Methodennamen
* in den Klammern steht eventuell zu verwendende Parameter an



* Methode beginnt mit main
* Endet mit Methodenrumpfes
* newBruch 🡪 Zugriff auf die Datei mit der Definition der Klasse Bruch
* Klasse Bruch wird im Hauptspeicher angelegt und über den Variablennamen b
* mehrere Objekte der gleichen Klasse mit new erzeugen, dabei werden sie nur ein eiziges mal erzeugt
* mit System.out.print wird die Methode ausgeführt
* beliebig viele (sowie gleiche) Methoden kann ein Block enthalten

Aufruf einer Methode erfolgt in drei Schritten

* Der aufrufende Block wird unterbrochen
* Der Methodenrumpf wird ausgeführt
* Der aufrufende Bloch wird mit der Anweisung nach dem Aufruf fortgesetzt
* Methodenrumpf stellt einen Block dar
* innerhalb sind beliebe Anweisungen, auch lokale Variablen
* alles nur innerhalb des Methodenrumpfes gültig
* Ein Bruch kann mit dem größten gemeinsamen Teiler (ggT)
* innerhalb eines Methodenrumpfes desselben festgelegten Objekts ist keine ausführlich Schreibweise für einen Elementzugriff notwendig
* (b.zaehler) außerhalb von einem Anwendungsprogramm muss man den Objektnamen mit angeben
* es ist möglich dass eine Methode eine andere Methode der Klasse aufruft

b.kuerzen();

b.ausgeben();

kann dadurch ersetzt werden:

b.gekuerztausgeben

*Abgrenzung von Bezeichnern*

* lokale Variablen und Datenelementen
* lokale Variablen wird innerhalb einer Methode definiert

void ausgeben();

int zaehler = 0; 🡪 namensgleiche lokale Variable

* Selbstreferenz ist die Bezeichnung für die Variable this und verweist immer auf das eigene Objekt
* wird ausgegeben mit: System.out.print(this.zaehler + “/“ + nenner);

*Werte übergeben*

**Methoden mit Parameter**

* Wird kein Übergabeparameter verwendet bleibt die klammer hinter dem Methodennamen leer
* es können in die Klammer Platzhalter zur Übergebe an die Methode eingetragen werden

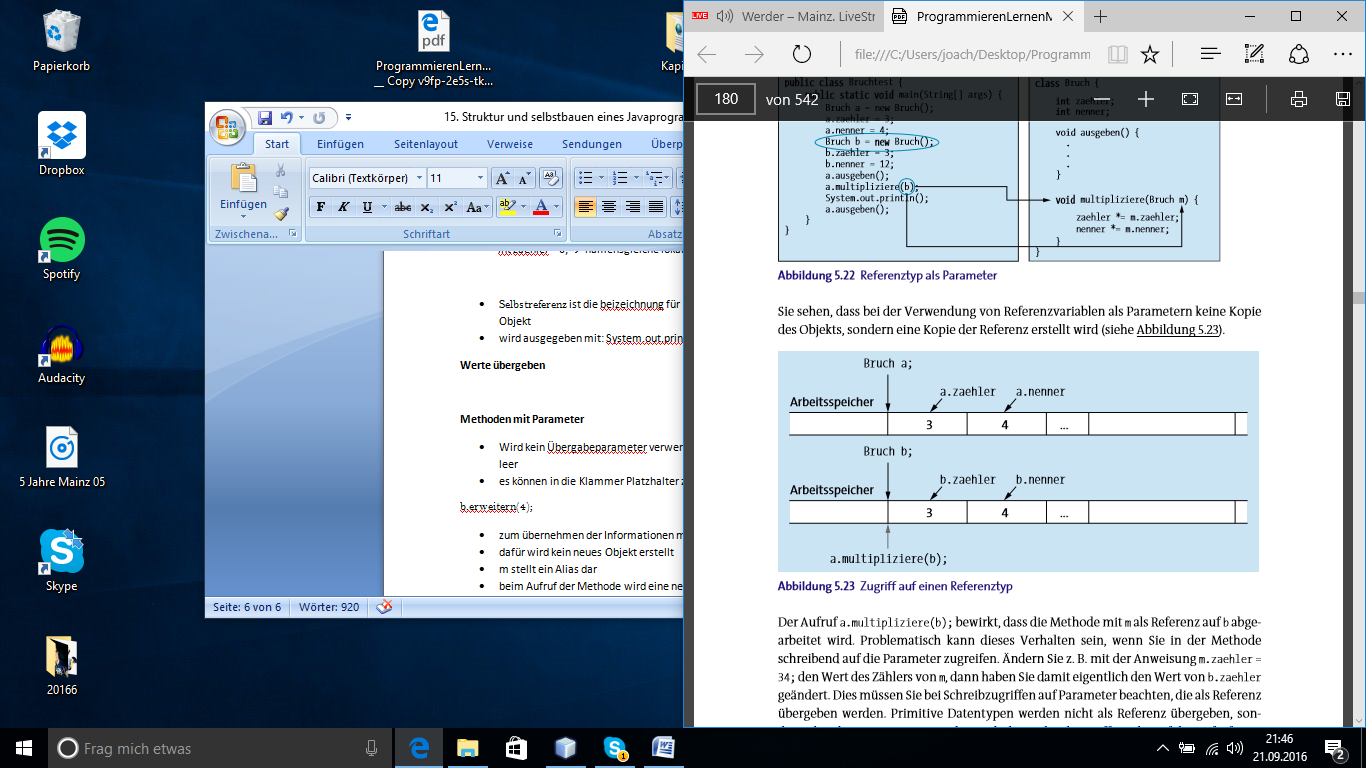
b.erweitern(4);

* zum übernehmen der Informationen muss ein entsprechender Behälter vorgesehen
* dafür wird kein neues Objekt erstellt
* m stellt ein Alias dar
* beim Aufruf der Methode wird eine neue Referenz (ein neuer Verweis) erzeugt

void erweitern(int a) {

zaehler \*= a;

nenner \*= a;



Primitive Daten werden nicht als Referenz übergeben, sondern als echte Kopie.

*Überladen von Methoden*

* es können mehrere Methoden mit gleichen Namen in einer Klasse existieren
* zur Unterscheidung gibt es die Parameterlister
* das benennen mehrerer Methoden die gleiche Namen haben bezeichnet man als Überladen
* bietet sich an wenn man die Methoden für ähnliche Funktionen benötigt

void setze(int z) {

zaehler = z;

nenner = 1;

void setze(int z; int n) {

zaehler = z;

nenner = n;

* Die Auswahl einer überladenen Methode ist für Compiler nicht immer eindeutig
* Compiler geht bei der Overload-Resolution folgend vor:
  + alle passenden Methoden werden gesammelt auch impliziete Typumwandlungen
  + bleibt eine Methode übrig wird diese Ausgewählt
  + passt kein Methode dann wird der Aufruf Fehlerhaft oder nicht übersetzt

*Ergebnisse:*

* Methode rechnet aus den Parametern einen Ergebniswert, der an die aufgerufene Methode zurückgeliefert werden soll

**Methoden mit Ergebnisrückgabe**

* Vor dem Methodennamen wird anstelle von void der Typ des Ergebnisses angegeben
* Im Rumpf der Methode steht eine return-Anweisung, die einen Ausdruck enthält, der dem Typ des Ergebnisses entspricht

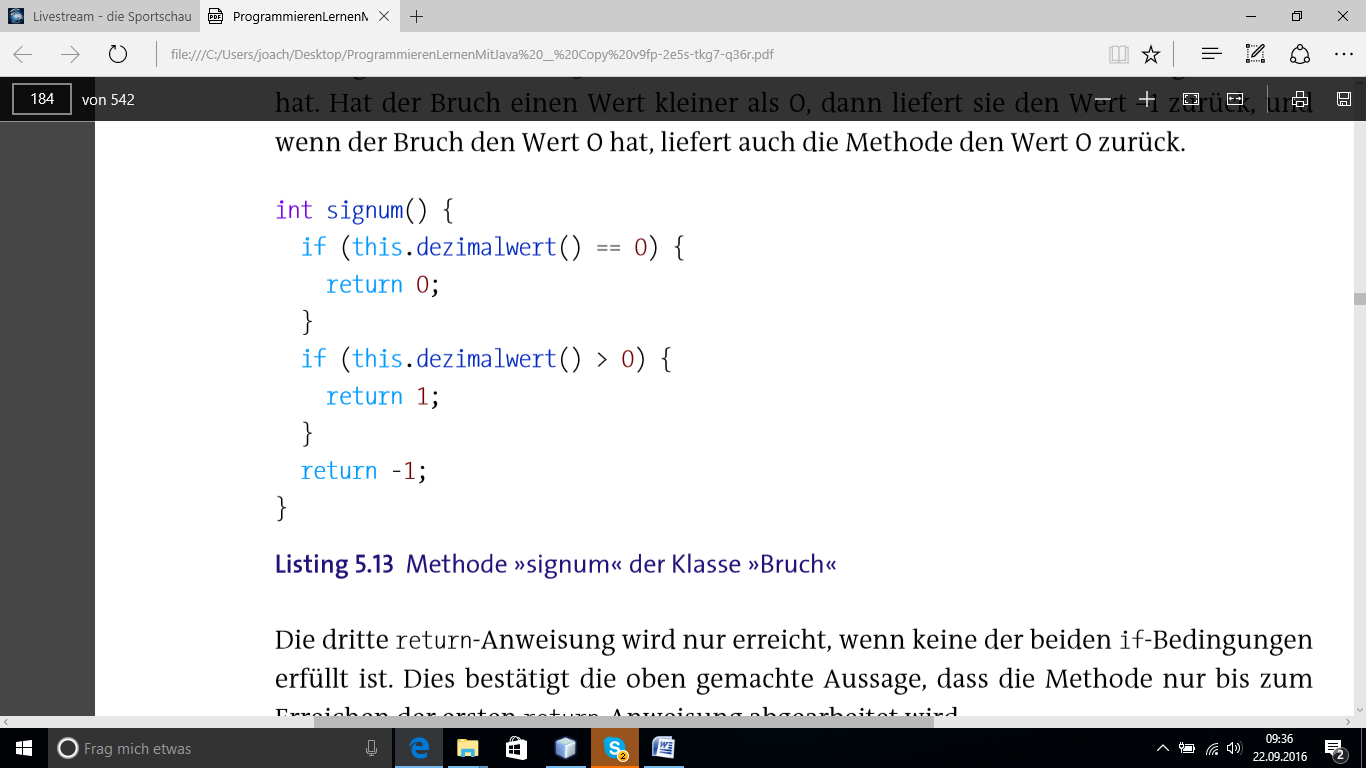
**Die 3 return-Anweisungen:**

1. return gibt den Wert an, der von der Methode zurückgegeben wird

* print-Anweisung ruft die Methode dezimalwert auf, die keine Parameter benötigt
* als Ergebnis wird ein double-Wert zurückgeliefert, der von der print-Anweisung ausgegeben wird

1. mehrere return-Anweisungen in einer Methode möglich

* nach Programmlogik entscheidet die zuerst erreichte return –Anwendung
* Methode signum mit einem Bruch kleiner als 0 liefert den Wert -1 zurück
* Methode signum mit einem Bruch der den Wert 0 hat liefert den Wert 0 zurück



* wenn keine der beiden if-Bedingungen erfüllt sind, bestätigt dies die oben gemachte Aussage, dass die Methode nur bis zum Erreichen der ersten return-Anweisung abgearbeitet wird
* es muss in jedem Fall eine return-Anweisung erreicht werden
* wird im Beispiel signum-Methode die letzte return-Anweisung auskommentiert wird ein Feler gemeldet, dass die Methode einen int-Wert zurückliefern muss

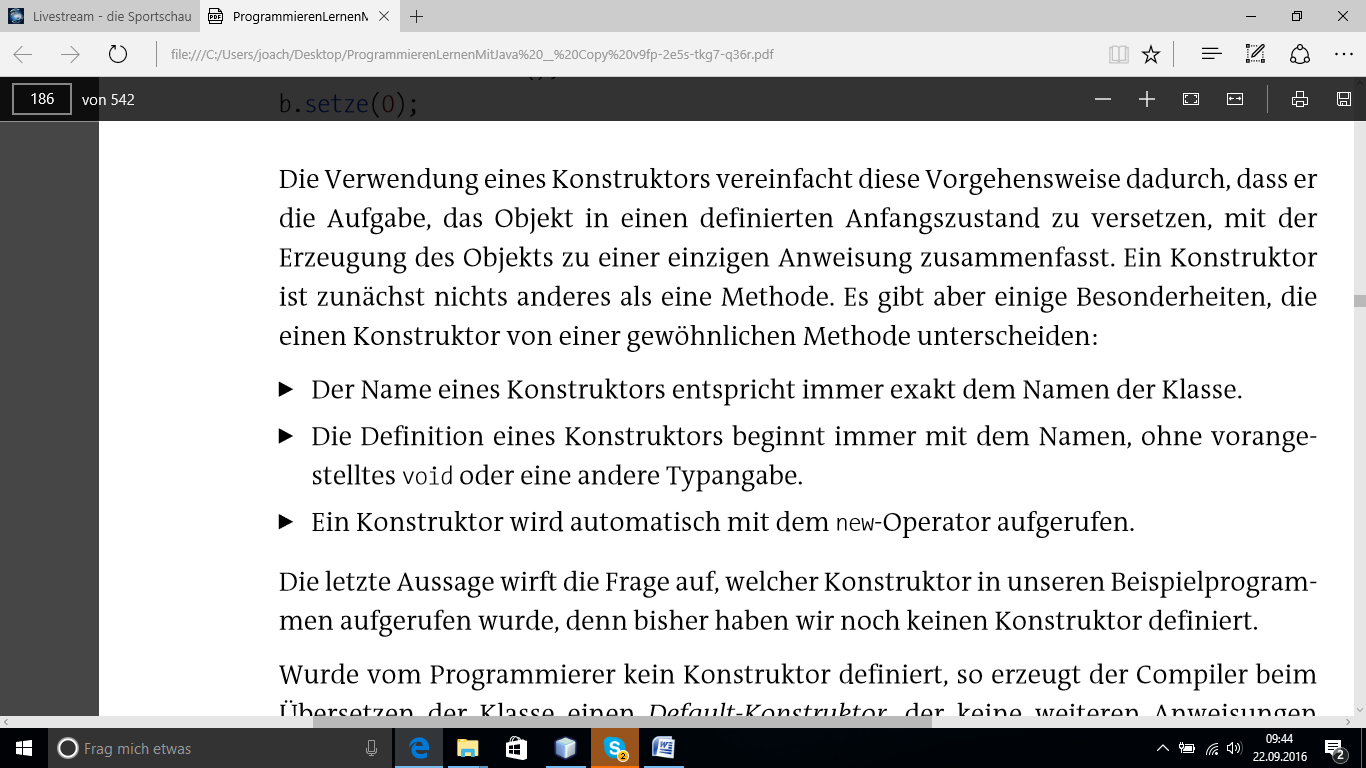
**Methoden ohne Ergbnisrückgabe**

* soll kein Ergebnis zurückgeliefert werden, wird der Ergebnistyp void angegeben
* ist ein Pseudo-Typ und steht für „nichts“
* mit return kann man ohne Ergebnisausdruck zur aufrufenden Anweisung zurückkehren
* am Ende einer Methode ohne Ergebnisrückgabe kehrt der Programmablauf automatisch zur aufgerufenen Anweisung zurück
* retur-Anweisung kann komplett fehlen, sie wird am Ende der Methode impliziert ergänzt
* überladene Methoden unterscheiden sich durch den Ergebnistyp

*Konstruktoren als spezielle Methoden*

* Wert des Objektes mit der Methode setze auf einen definierten Wert
* Ein Konstruktor ist nichts anderes als eine Methode

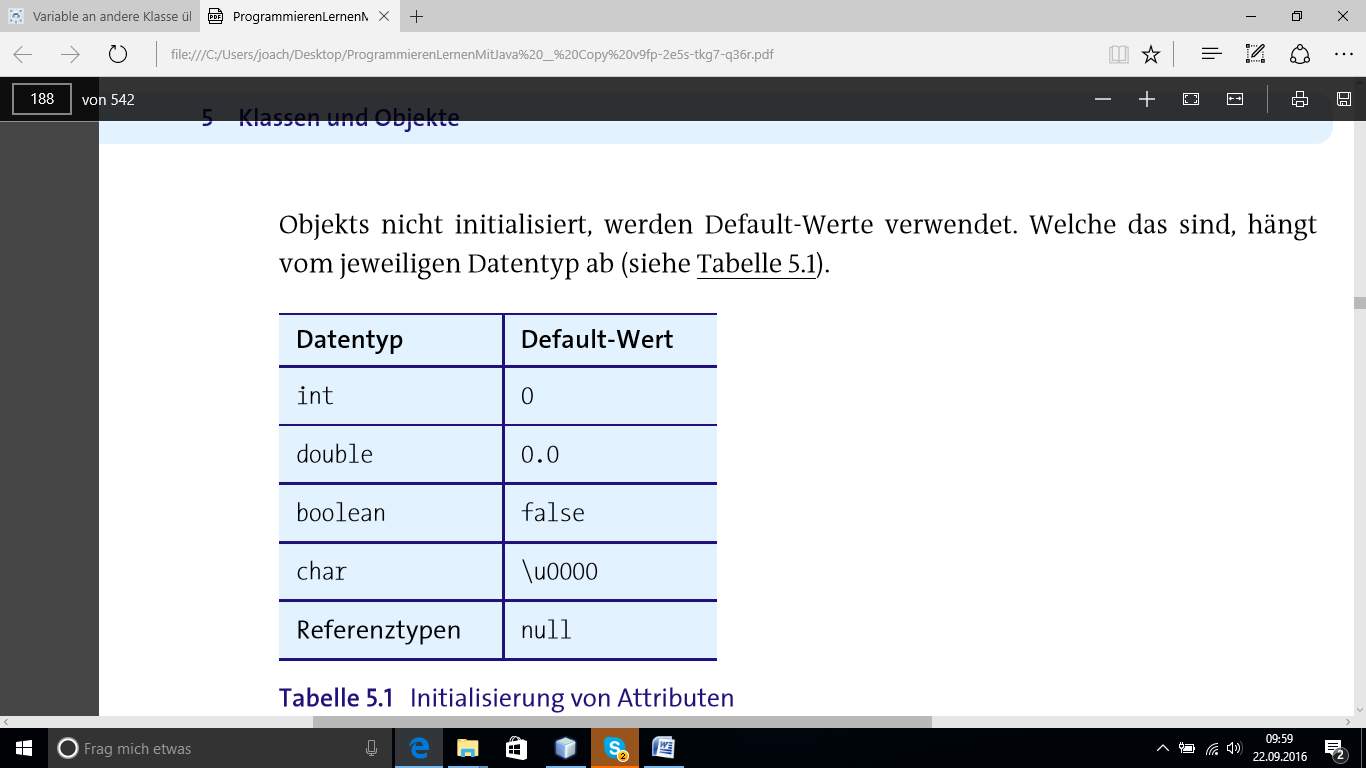
Unterschiede zwischen Konstruktor und Methoden:



* wird kein Konstruktor definiert, erzeugt der Compiler beim übersetzen der Klasse einen Default-Konstruktor
* jede Klasse besitzt einen Konstruktor
* Konstruktor besitzt den gleichen Namen wie die Klasse
* Klasse 🡪Bruch() Konstruktor 🡪 Bruch b = new Bruch();
* Konstruktor wird genauso definiert wie eine andere Methode und ohne vorangestelltes void

*Konstruktoren mit Parametern*

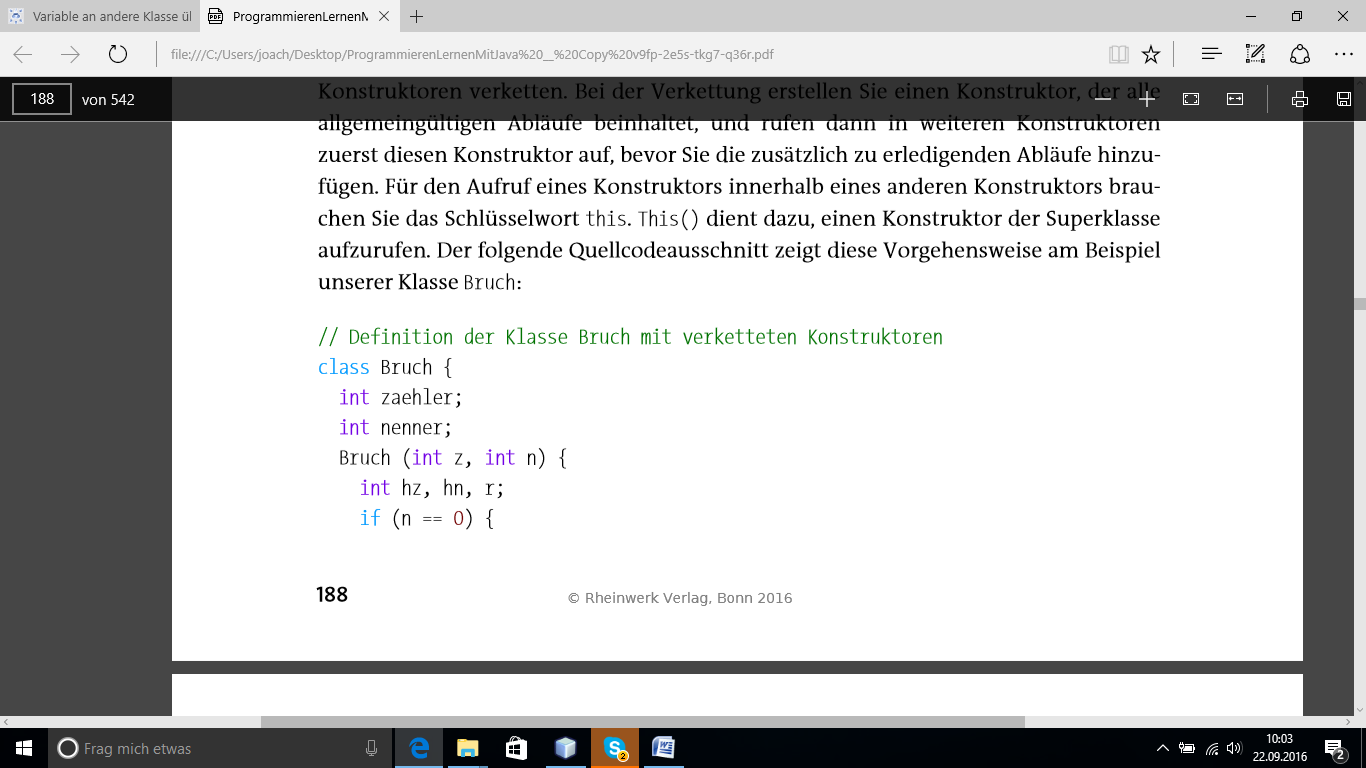
* es können beliebig weitere Konstruktoren mit Parametern definiert werden
* bei erzeugen eines Objekts wird immer der passende Konstruktor von der Parameterliste verwendet
* Konstruktoren mit Parameter heißen Custom-Konstruktoren
* die Attribute eines Objekts werden nicht initialisiert
* werden Attribute (Datenelemente) eines Objekte nicht initialisiert werden Default-Werte verwendet

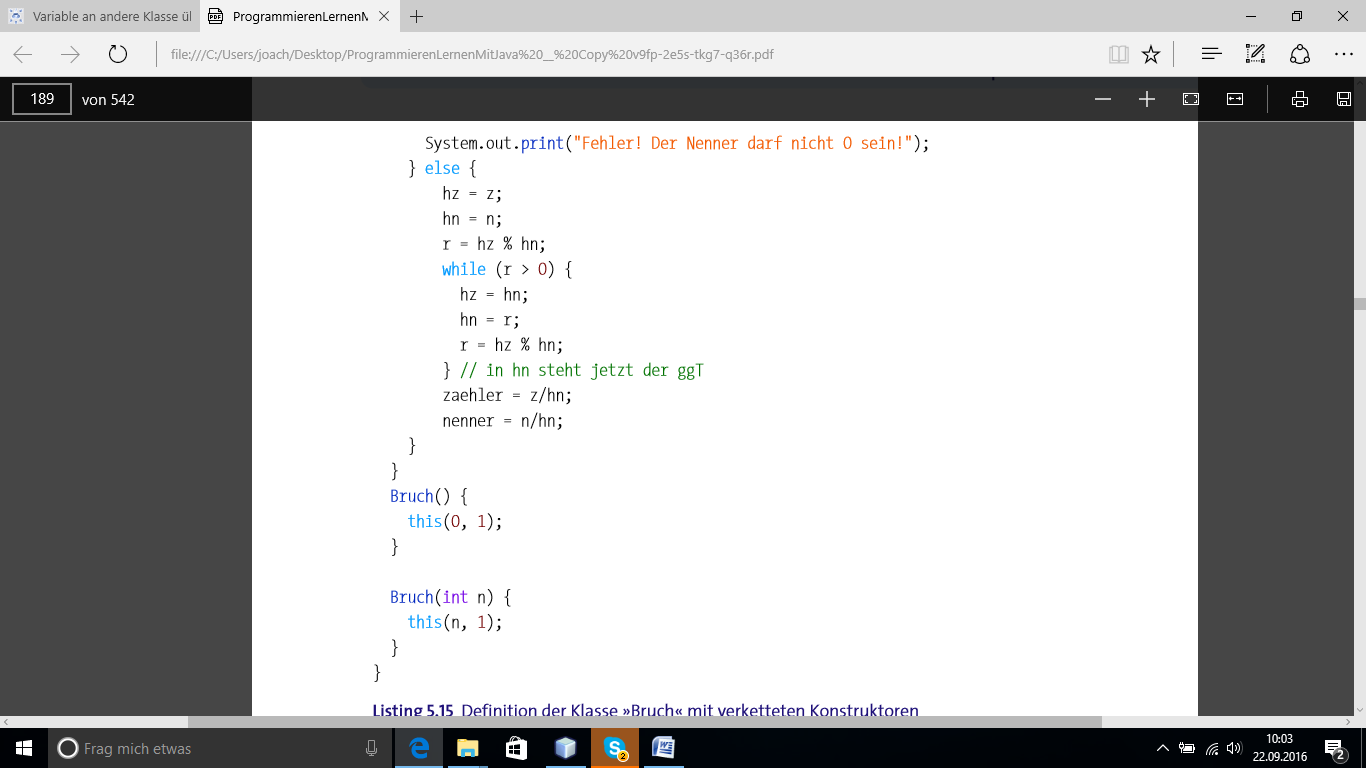


* Wurden die Attribute einer Klasse mit der Definition breits initialisiert überschreib der anschließende Konstruktorablauf wieder diese Werte.

*Verketten von Kontruktoren*

* in Konstruktoren finden Überprüfungen statt, die in aufwendigen Kontrollstrukturen stattfinden
* dazu werden die Konstruktoren verkettet
* dazu erstellt man einen Konstruktor der alle allgemeingültigen Abläufe beinhaltet
* für Aufruf eines Konstruktors innerhalb eines anderen Konstruktor verwendet man das Schlüsselwort this. This()





* Bruch mit dem Wert Null (Zähler = 0 und Nenner = 1)
* dies übernimmt der erste Konstruktor, wird über die Anweisung this(0, 1); aufgerufen
* ebenso wird mit dem zweiten Konstriktor verfahren, dieser erwartet ein Parameter n mit ganzzahligen Wert also n/1, initialisieren
* wird auch dem ersten Konstruktor this(n,1); übertragen
* ein verketteter Konstruktor mit this muss immer als erste Anweisung im Konstruktorrumpf stehen

**Notiz: Brotcrunsher** 🡪 **youtube.de Tutorial 10**