**Methoden**

* Eine Erweiterung
* Können in Klassedn neben den Datenelementen definiert werden
* Beschrieben das Verhalten einer Klasse

**Aufbau von Methoden :**

**Beispiel:**

Allgemeiner Aufbau:

Rückgabetyp Methodenname (Parametertyp, Parameter1, …)

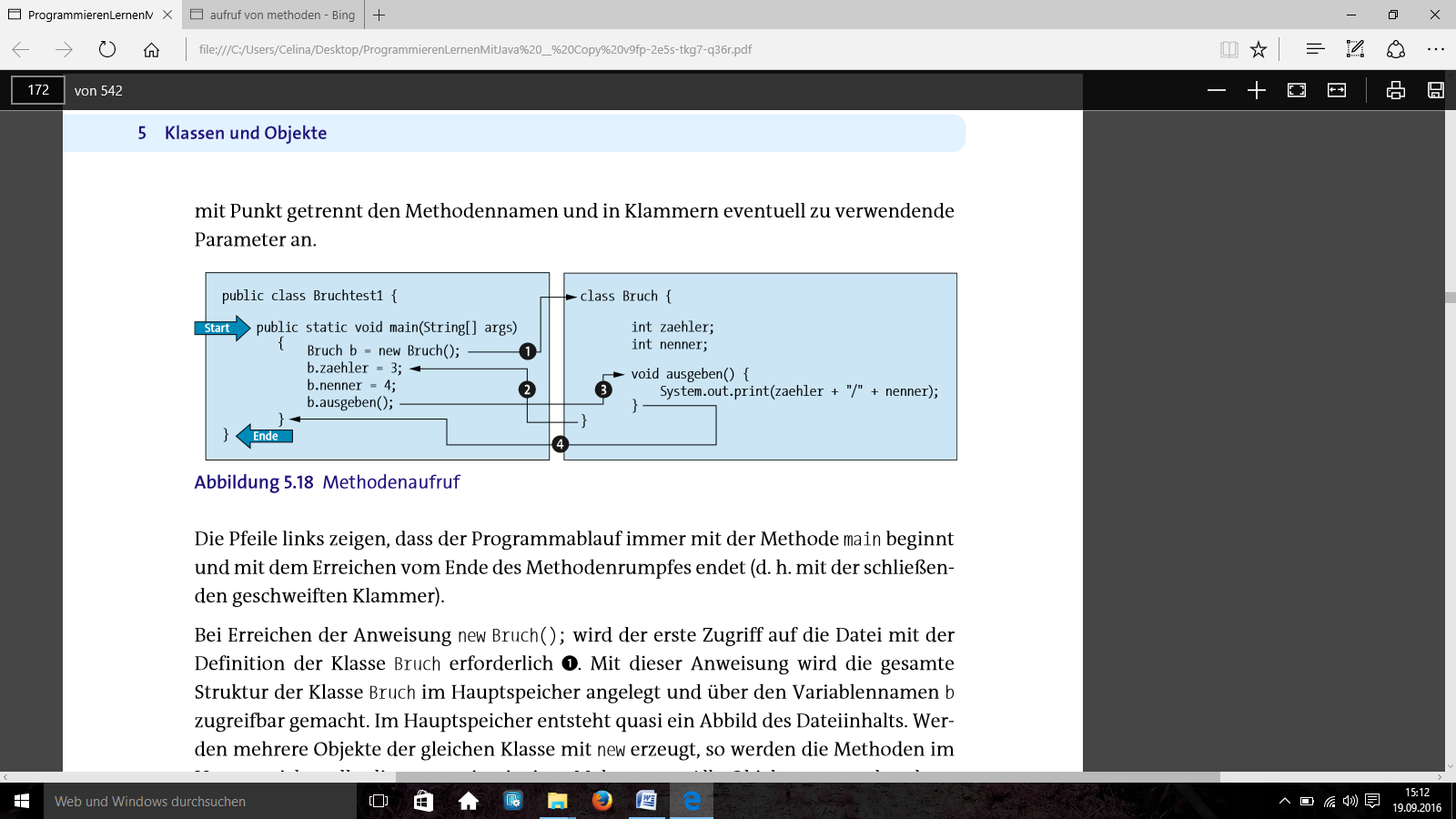
Definieren einer Methode, die dafür sorgt, dass ein Objekt der Klasse *Bruch* in einem bestimmten Format ausgegeben wird:

void ausgeben () { // Kopf der Methode

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

}

**Aufruf von Methoden:**

****

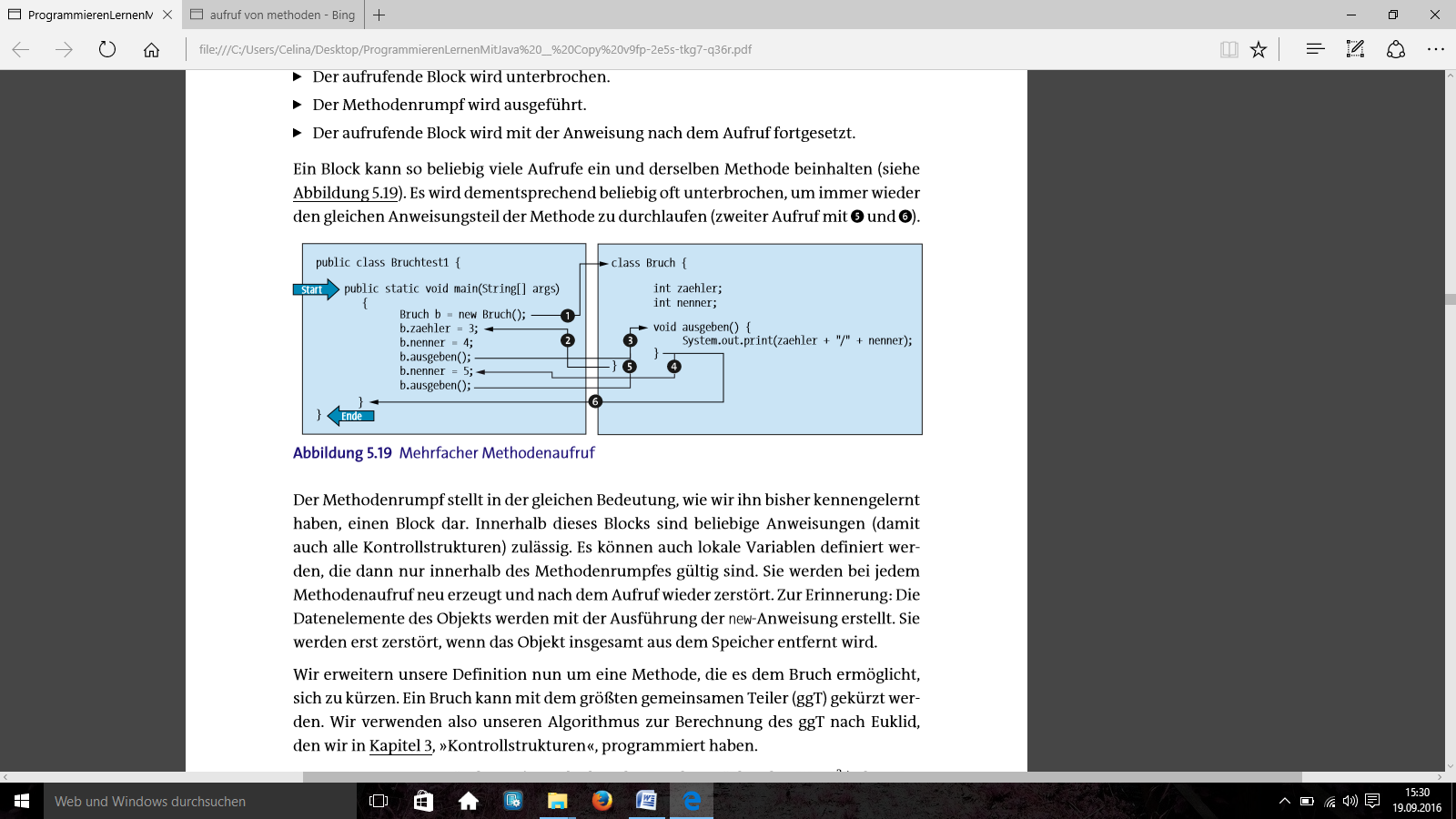
* Programmablauf beginnt immer mit der Methode main und mit Erreichen vom Ende des Methodenrumpfs endet

1. *newBruch();* = erster Zugriff auf die Datei mit der Definition der Klasse *Bruch* erforderlich
   1. gesamte Struktur der Klasse *bruch* wir im Hauptspeicher angelegt und über den Variablenname *b* zugreifbar gemacht
   2. werden mehrere Objekte der gleichen Klasse mit *new* erzeugt, so werden die Methoden im Hauptspeicher nur ein einziges mal erzeugt
   3. alle Objekte verwenden dann die gleiche Implementierung der Methode
2. Nächste Anweisung im Programm wird abgearbeitet
   1. Dem Zähler unseres Bruchs wird der wert 3 und dem Nenner 4 zugewiesen
3. *b.ausgeben();* = Zugriff auf Abbild im Hauptspeicher
   1. Methode *ausgeben()* des Onjekts Bruch wird abgearbeitet
   2. In diesem Fall ausführung der Anweisung *System.out.print*
4. Nach erreichen des Methodenrumpfes, wird mit der nächsten Anweisung im Programm fortgefahren
   1. Ende des Methodenrumpes *main* erreicht, und Programm beendet

**Mehrfacher Methodenaufruf**

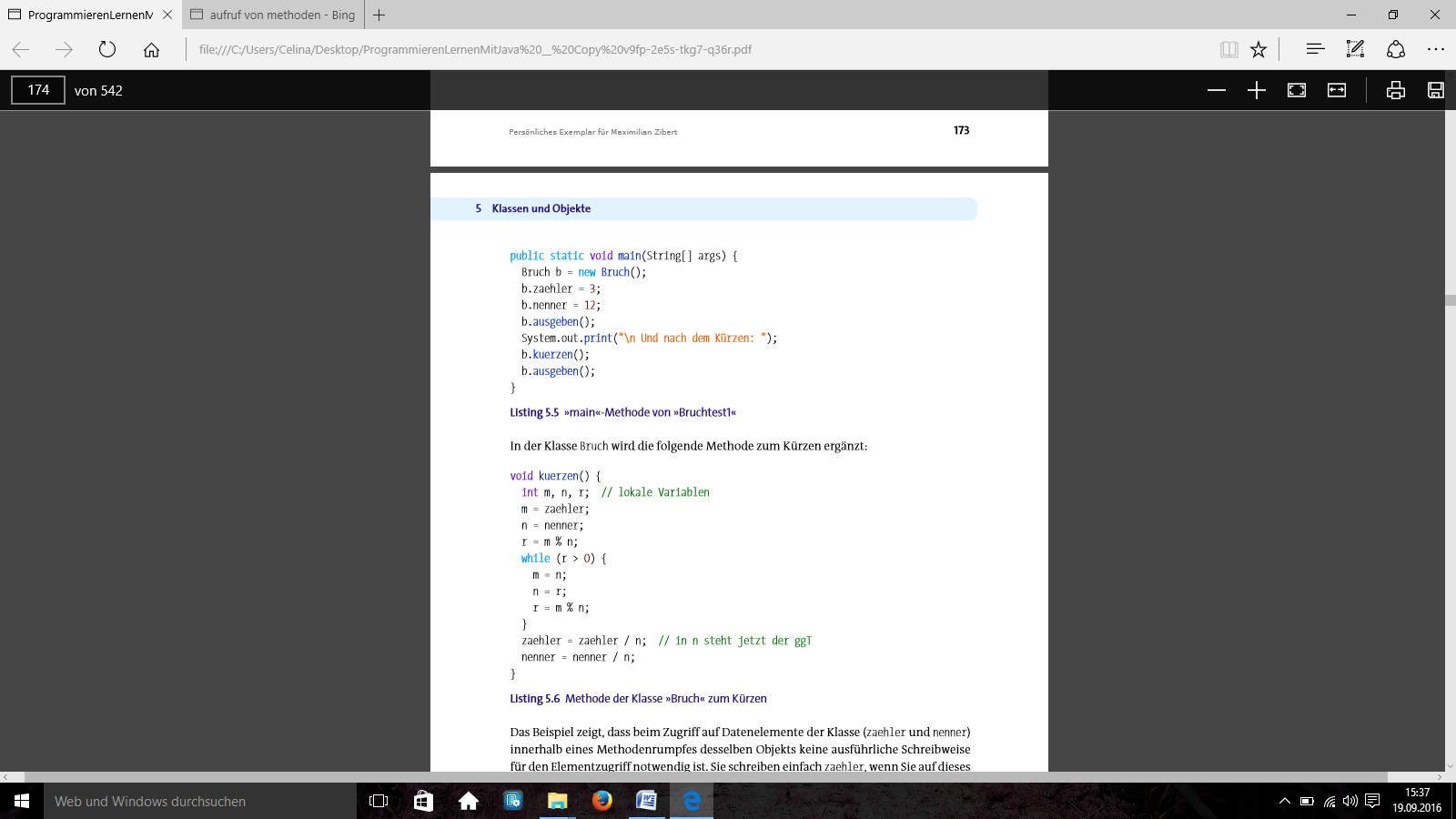
**Aufruf einer Methode in 3 Schritten:**

* Der aufrufende Block wird unterbrochen
* Der Methodenrumpf wird ausgeführt
* Der aufrufende Block wird mit der Anweisung nach dem Aufruf fortgesetzt



* Methodenrumpf stellt einen Block dar
* Innerhalb deses Blocks sind verschiedene anweisungen zulässig
* Erinnerung : Die Datenelemente des Objekts werden mit der Ausführung der *new*-Anweisung erstellet
  + Sie werden erst zerstört, wenn das Objekt insgesamt aus dem Speicher entfernt wird
* Erweiterung der Definition um eine Methode
  + Ermöglicht dem Bruch sich zu kürzen
  + Verwenden des Algorithmus zur Berechnung des ggT nach Euklid (Kapitel 3)
* Testprogramm wird geändert
  + Bruch mit dem Wert 3/12
  + Bruch wird ungekürzt ausgegeben, gekürzt umd schließlich nocheinmal gekürzt ausgegeben

**Beispiel code:**



* Zeigt, dass beim Zugriff auf Datenelemente der Klasse (*zaehler* und *nenner*) innerhalb eines Methodenrumpfes desselben Objekts keine ausführliche Schreibweise für den Elementzugriff notwendig ist

**Ergänzung, die einen Bruch gekürzt ausgibt:**

void gekuerztausgeben () {

kuerzen();  
 ausgeben();

}

* Zurückgreifen auf die bereits definierten Methoden *kuerzen()* und *ausgeben()*

🡪 b.kuerzen();

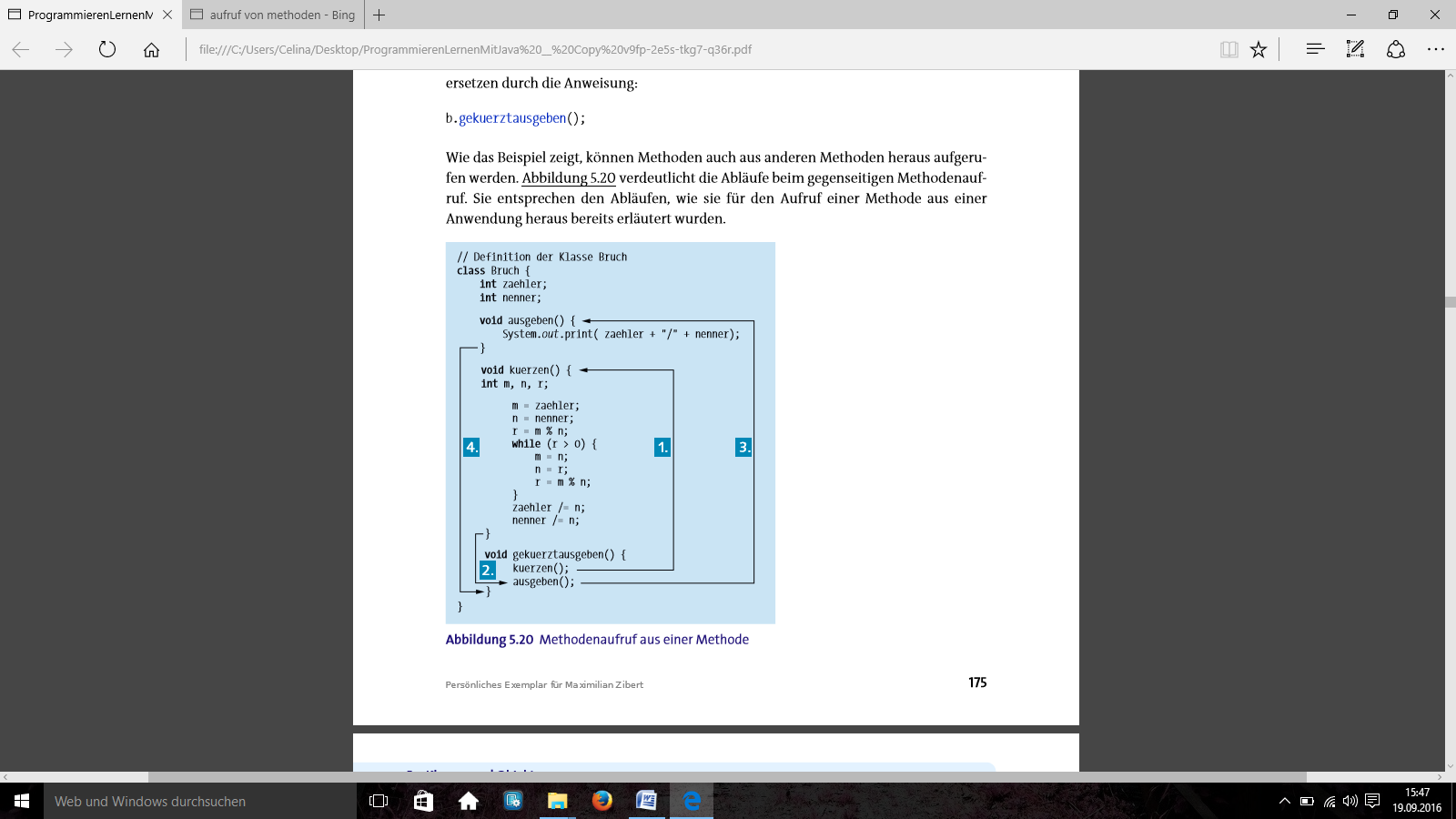
b. ausgeben();

können durch die Anweisung

b.gekuerztazsgeben();

ersetzt werden.

**Methodenaufruf aus einer Methode:**



**Abgrenzung von Bezeichnern**

* In der Methode *ausgeben()* wird eine lokale Variable mit dem Namen *zaehler* definiert
  + Compiler zeigt keine Fehlermeldung

**Beispiel:**

void ausgeben() {

int zaehler = 0; //namensgleiche lokale Variable

System.out.print(zaehler + “/“ + nenner);

}

Frage: Worauf greift die Methode *ausgeben()* zurück?

* eine Variable mit dem Namen *this* existiert, um an ein verdecktes Datenelement einer Klasse heran zu kommen
  + *this* = Selbstreferenz
* *this* verweist auf das Objekt *b*, wenn Anweisung umgeändert wird:

System.out.print(this.zaehler + “/“ + nenner);

* der Wert des Datenelements *zaehler* wird mit dem Wert 3 ausgegeben

**Methoden mit Parameter**

* Keine Übergabeparameter = leere Klammer hinter Methodenname
* In der Klammer können platzhalter für zur übergabe an die Methode eingetragen werden

b.erweitern(4);

* Um Info zu übernehm,en muss behälter umgeändert werden
* Behälter für ganze Zahlen wird benötigt

void erweitern(int a) {

zaehler \*= a;

Nenner \*= a;

}

Schritte bei Verwendung mit Parameter:

* Die Werte aller Argumente werden berechnet
* Die Parameter werden angelegt
* Die Argumentwerte werden an die Parameter übergeben
* Der aufrufende Block wird unterbrochen
* Der Methodenrumpf wird abgearbeitet
* Die Parameter werden wieder zerstört
* Der aufrufende Block wird fortgesetzt

**Referenztypen als Parameter**

Referenztypen = Objekte

* Methode mit einem Bruch mit einem als Parameter übergebenen Bruch multiplizieren

**Beispiel**:

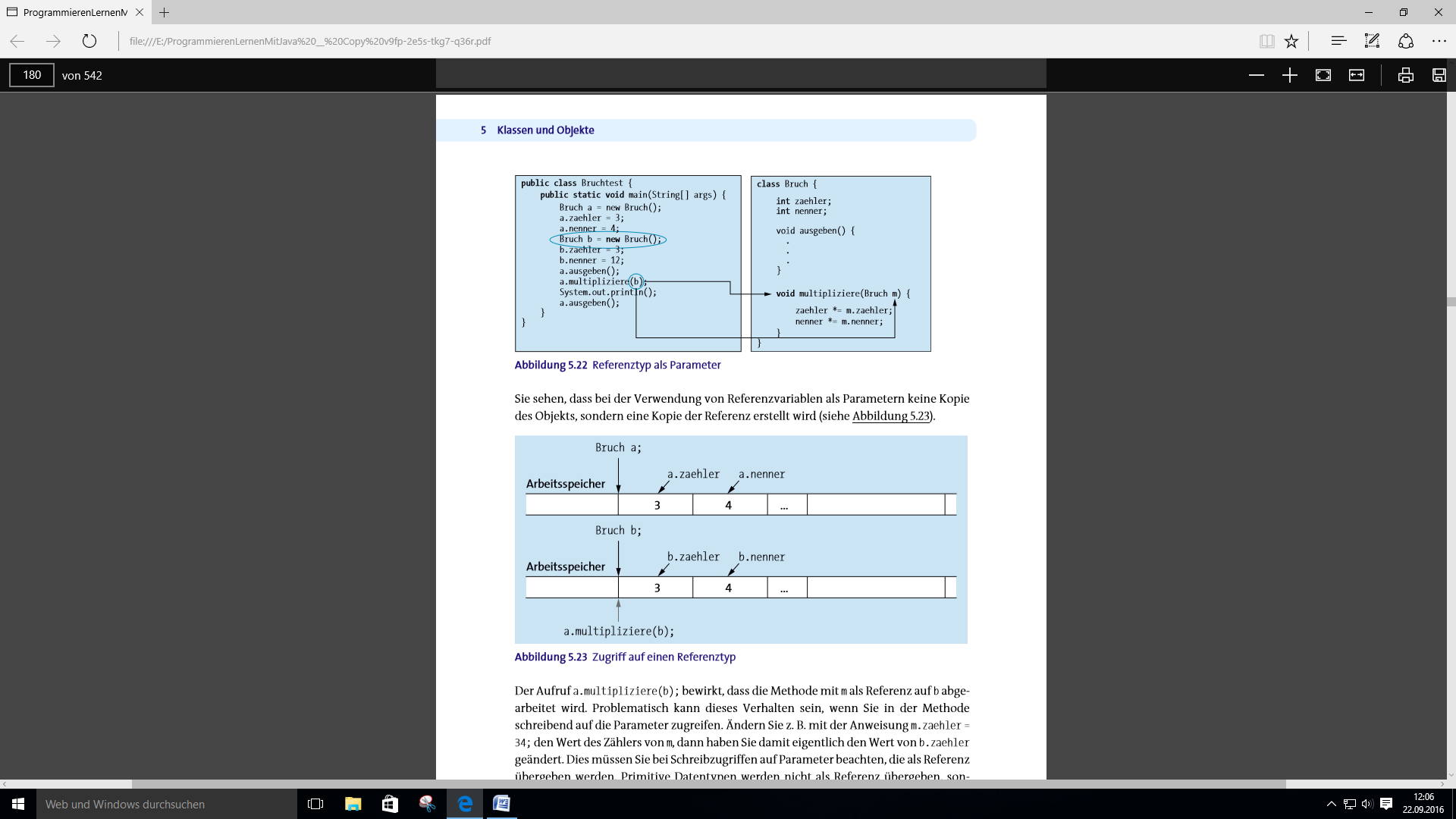
void multiplizieren(Bruch m) {

zaehler \*= m.zaehler;

Nenner \*= m.nenner;

}

* *b* = *new*
* Beim Aufruf auf die Methode nur eine neue Referenz auf das Argument mit dem Namen *m* erzeugt
* *m* greift auf das Original *b* zu



**Überladen von Methoden**

* innerhalb einer Klasse können mehrere Methoden mit einem Namen existieren
* Überladen bedeutet: Eine Methode mit einem Namen erstellen, der bereits für eine andere Methode verwendet wurde
* Bietet sich an, wenn mehrere Methoden für eine ähnliche Funktion benötigt wird

**Beispiel:**

void setze(int z) {

zaehler \* z;

nenner = 1;

}

void setze(int z, int n) {

zaehler = z;

nenner = n;

}

**Methoden mit Ergebnisrückgabe**

* In Klasse *Bruch* kann eine Methode erstellt werden, die den Wert des bruchs als Dezimalzahl zurückliefert
  + Vor Methodenname wird anstelle von *void* der Typ des Ergebnisses angegeben
  + Rumpf der Methode steht eine *return*-Anwendung

**Beispiel**:

double dezimalwert() {

return (double) zaehler/nenner;

}