



Einsendeaufgabe Typ B

Grundlagen der objektorientierten Programmierung

| | | | |
|---|-----------------------|----------------------------|--|
| Name: | | Vorname: | Einsendeaufgabencode: B-GOPB01-XX2-K04 |
| Straße: | | PLZ, Ort: | Korrektor: |
| Matrikelnummer: | | Studiengangsnummer: | Datum: |
| Name der B-Aufgabe: B-G O P B 0 1 X X | Variante: 2 | Auflage: 0722K04 | Note: |
| Bezogene Studienhefte: JAPR01, JAPR02, PYTH00, PYTBU | | | Unterschrift: |

Bitte reichen Sie Ihre Lösungen über den Online-Campus ein. Falls Sie uns diese per Post senden wollen, dann fügen Sie bitte die Aufgabenstellung und den Einsendeaufgabencode hinzu.

1. Tree (Python)

Das Tool „tree“¹ stellt den Inhalt des aktuellen Verzeichnisses in einer baumartigen Struktur dar. Falls „tree“ auf ein Unterverzeichnis stößt, werden die Dateinamen in diesem Verzeichnis eingerechnet. Dies setzt sich für Unter-Unterverzeichnisse usw. rekursiv fort.

Ein Beispiel für die Ausgabe von „tree“ mit einem Verzeichnis mit drei Unterverzeichnissen und einem Unter-Unterverzeichnis zeigt Abb. 1.1.

```
thomas@tower: ~/dev/java/javahefte/B-Aufgaben$ python3 python/tree.py
aufgaben.org
aufgaben.pdf
aufgaben.tex
java
├── aminoacids.csv
├── Aminoacids.java
├── KochKurve.java
├── Kurve.java
└── MazeGenerator.java
media
├── 600px-Kochkurve.jpg
├── koch0.png
├── koch1.png
├── koch2.png
├── koch5.png
├── kurve3.png
├── kurve4.png
├── kurve5.png
├── kurve6.png
├── maze.pdf
├── maze_10.png
├── maze_25.png
├── maze_5.png
├── tree.png
└── tree_simple.png
python
├── pycache
│   ├── caesar.cpython-37.pyc
│   └── paste.cpython-37.pyc
├── caesar.py
├── data.csv
├── names.txt
├── numbers.txt
├── paste.py
├── tree.py
└── Studheft.sty

4 directories, 32 files
thomas@tower: ~/dev/java/javahefte/B-Aufgaben$
```

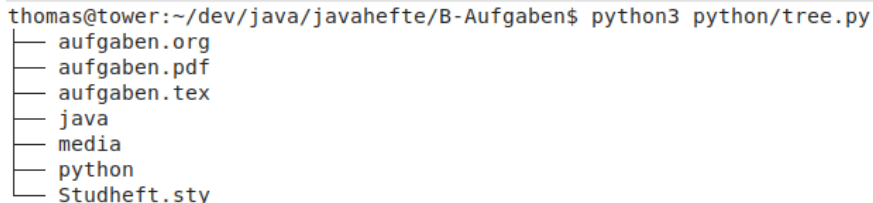
Abb. 1.1: Ausgabe des Programms „tree“

¹ siehe [https://en.wikipedia.org/wiki/Tree_\(command\)](https://en.wikipedia.org/wiki/Tree_(command))

In dieser Aufgabe sollen Sie eine (vereinfachte) Version von „tree“ in Python implementieren.

- a) Implementieren Sie eine Funktion `print_directory(path)`. Diese Funktion soll mittels `os.scandir` über die Dateien im übergebenen Verzeichnis iterieren und die Dateinamen wie in Abb. 1.2 gezeigt ausgeben. Im Hauptprogramm rufen Sie `print_directory` mit dem aktuellen Verzeichnis auf:

```
print_directory(".")
```



```
thomas@tower:~/dev/java/javahefte/B-Aufgaben$ python3 python/tree.py
├─ aufgaben.org
├─ aufgaben.pdf
├─ aufgaben.tex
├─ java
├─ media
├─ python
└─ Studheft.sty
-
```

Abb. 1.2: Ausgabe des Programms „tree“ *ohne* Rekursion

Sie benötigen die Unicode-Symbole mit der Dezimaldarstellung 9472, 9474, 9492 und 9500. Den waagerechten Strich erhalten Sie bspw. über `chr(9472)`. Beachten Sie die Ausgabe der letzten Datei im Verzeichnis!

Hinweise: Damit Sie die Dateien nach ihren Namen sortiert ausgeben können, übergeben Sie die Rückgabe von `os.scandir` an die Funktion `sorted`. Dem benannten Parameter `key` der Funktion `sorted` weisen Sie die folgende Lambda-Funktion zu:

```
lambda f: f.name.lower().
```

8 Pkt.

- b) Erweitern Sie die Funktion `print_directory` um einen zweiten Parameter für die Einrückungsebene:

```
print_directory(path, indentation_level=0)
```

Falls Sie auf ein Verzeichnis stoßen (verwenden Sie die `DirEntry`-Methode `is_dir`), rufen Sie `print_directory` rekursiv für dieses Verzeichnis und `indentation_level+1` auf. Passen Sie die Ausgabe anhand der aktuellen Einrückungsebene `indentation_level` so wie in Abb. 1.1 gezeigt an.

8 Pkt.

- c) Ergänzen Sie am Ende der Ausgabe die Zählung

```
X directories, Y files
```

Hinweis: Lassen Sie `print_directory` ein Tupel (`nfiles`, `ndirectories`) zurückliefern.

4 Pkt.

Σ 20 Pkt.

2. Cäsar-Chiffre (Python)

Die Cäsar-Verschlüsselung ist ein einfaches und sehr unsicheres Verfahren zur Verschlüsselung eines Textes. Jeder Buchstabe wird um eine durch den Schlüssel bestimmte Anzahl Buchstaben im Alphabet zyklisch verschoben.

Beispiel: Angenommen, der Schlüssel sei 3. Der Klartext

Das ist ein Text.

wird dann verschlüsselt zu

Gdv lvw hlq Whaw.

- a) Implementieren Sie eine Python-Funktion `encode_text(text, key)`, die den übergebenen Text nach dem Cäsar-Verfahren verschlüsselt und den verschlüsselten Text zurückliefert.

Zum Entschlüsseln rufen Sie die Funktion `encode_text` mit dem negativen Schlüssel auf:
`encode_text("Gdv lvw hlq Whaw.", -3)`

Alternativ kann `encode_text` zum Entschlüsseln auch mit 26-key aufgerufen werden (das Alphabet hat 26 Buchstaben):

`encode_text("Gdv lvw hlq Whaw.", 26-3)`

Hinweise: Verwenden Sie die String-Methoden `isupper`, `islower` und die Funktionen `ord` und `chr`. Nur Buchstaben sollen verschlüsselt werden. Sonstige Zeichen werden unverändert in den verschlüsselten Text übernommen.

Sie können davon ausgehen, dass der Text nur ASCII-Zeichen (also keine Umlaute, kein ß) enthält.

Implementieren Sie zudem ein Hauptprogramm (`main`), sodass das Python-Skript wie folgt auf der Konsole aufgerufen werden kann:

`python caesar.py "Das ist ein Text" 3`

Wenn weniger als zwei Parameter übergeben wurden, soll das Skript mit einer Fehlermeldung abbrechen.

10 Pkt.

- b) Implementieren Sie eine Python-Funktion `string_histogram(text)`, die ein Dictionary zurückliefert, das für jeden vorkommenden Buchstaben die Häufigkeit im Text zählt. Für das Beispiel

`string_histogram("Das ist ein Text")`

ergibt sich:

`{'d': 1, 'a': 1, 's': 2, 'i': 2, 't': 3, 'e': 2, 'n': 1, 'x': 1}`

Hinweis: Verwenden Sie die String-Methoden `isalpha` und `lower`.

5 Pkt.

- c) Nun möchten wir ermitteln, mit welcher Frequenz bzw. Wahrscheinlichkeit ein bestimmter Buchstabe im Text vorkommt. Implementieren Sie hierzu eine Python-Funktion

`frequencies(histogram)`

die für ein übergebenes String-Histogramm eine Liste der Länge 26 mit den zugehörigen Wahrscheinlichkeiten zurückliefert („Wahrscheinlichkeitsvektor“).

Für den Aufruf

```
frequencies(string_histogram("Das ist ein Text"))
```

ergibt sich beispielsweise:

```
[0.07692307692307693, 0, 0, 0.07692307692307693,
0.15384615384615385, 0, 0, 0, 0.15384615384615385, 0, 0, 0, 0,
0.07692307692307693, 0, 0, 0, 0, 0.15384615384615385,
0.23076923076923078, 0, 0, 0, 0.07692307692307693, 0, 0]
```

(der Buchstabe 'a' kommt mit 7,69 % Wahrscheinlichkeit im Text vor, der Buchstabe 'b' kommt gar nicht im Text vor usw.)

5 Pkt.

- d) Zuletzt möchten wir die Cäsar-Verschlüsselung knacken. Hierzu bestimmen wir anhand eines bekannten, längeren Beispieltexts die Wahrscheinlichkeiten, wie oft ein einzelner Buchstabe des Alphabets in diesem Text vorkommt. Danach probieren wir für den verschlüsselten Text alle 26 möglichen Schlüssel durch und vergleichen die Buchstabenfrequenzen mit den Frequenzen des Beispieltexts. Der Schlüssel mit dem kleinsten „Abstand“ zum Beispieltext entschlüsselt mit großer Wahrscheinlichkeit den Text.

Implementieren Sie hierzu eine Python-Funktion

```
crack_caesar(exampletext, text)
```

die den Code knackt und den Klartext zum verschlüsselten Argument `text` zurückliefert.

Für den Vergleich der Wahrscheinlichkeitsvektoren eignet sich die Funktion χ^2 („Chi-squared“):

$$\chi^2 = \sum_{i=1}^n \frac{(O_i - E_i)^2}{E_i}$$

Hierbei bezeichnet n die Anzahl der Buchstaben im Alphabet, E den Wahrscheinlichkeitsvektor des Beispieltexts und O den Wahrscheinlichkeitsvektor des verschlüsselten Texts. Je kleiner χ^2 , desto ähnlicher sind sich die beiden Wahrscheinlichkeitsvektoren.

Als Beispieltext können Sie den folgenden Ausschnitt aus dem Stück „Julius Caesar“ von William Shakespeare verwenden:

I know that virtue to be in you, Brutus, As well as I do know your outward favour. Well, honour is the subject of my story. I cannot tell what you and other men Think of this life; but, for my single self, I had as lief not be as live to be In awe of such a thing as I myself. I was born free as Caesar; so were you: We both have fed as well, and we can both Endure the winter's cold as well as he: For once, upon a raw and gusty day, The troubled Tiber chafing with her shores, Caesar said to me 'Darest thou, Cassius, now Leap in with me into this angry flood, And swim to yonder point?' Upon the word, Accoutred as I was, I plunged in And bade him follow; so indeed he did. The torrent roar'd, and we did buffet it With lusty sinews, throwing it aside And stemming it with hearts of controversy; But ere we could arrive the point proposed, Caesar cried 'Help me, Cassius, or I sink!' I, as Aeneas, our great ancestor, Did from the flames of Troy upon his shoulder The old Anchises bear, so from the waves of Tiber Did I the tired Caesar. And this man Is now become a god, and Cassius is A wretched creature and must bend his body, If Caesar carelessly but nod on him. He had a fever when he was in Spain, And when the fit was on him, I did mark How he did shake: 'tis true, this god did shake; His coward lips did from their colour fly, And that same eye whose bend doth awe the world Did lose his lustre: I did hear him groan: Ay, and that tongue of his that bade

the Romans Mark him and write his speeches in their books, Alas, it cried 'Give me some drink, Titinius,' As a sick girl. Ye gods, it doth amaze me A man of such a feeble temper should So get the start of the majestic world And bear the palm alone.

Bestimmen Sie anhand der Funktion `crack_caesar` sodann den Klartext zu dem folgenden verschlüsselten Text:

Reu jf zk zj. Wfi kyzj kzdv Z nzcc cvrmv pfl: Kf-dftifn, zw pfl gcvrjv kf jgvrb nzky dv, Z nzcc tfdv yfdv kf pfl; fi, zw pfl nzcc, Tfdv yfdv kf dv, reu Z nzcc nrzk wfi pfl.

10 Pkt.

Σ 30 Pkt.

3. Klammerprüfung (Java)

- a) In dieser Aufgabe sollen Sie ein Programm implementieren, das einen übergebenen Quelltext auf eine korrekte Klammerung überprüft.

Wenn der Quelltext korrekt geklammert ist, soll die Ausgabe lauten:

Der Quelltext ist korrekt geklammert

Andernfalls geben Sie aus:

Der Quelltext ist nicht korrekt geklammert

Sie können zur Lösung der Aufgabe den gesamten Dateinhalt zunächst in einen String einlesen. Gehen Sie dann wie folgt vor: Iterieren Sie zeichenweise über den String. Wenn Sie auf eine öffnende Klammer, d. h. eines der Zeichen (, { oder [, treffen, legen Sie die Klammer auf einen Stack. Wenn Sie dagegen auf eine schließende Klammer treffen, d. h. eines der Zeichen), } oder], überprüfen Sie, ob oben auf dem Stack das entsprechende Gegenstück liegt. Wenn ja, entfernen Sie das oberste Zeichen vom Stack. Andernfalls brechen Sie ab – der Quelltext ist in diesem Fall nicht korrekt geklammert.

Der Quelltext ist dann korrekt geklammert, wenn der Stack am Ende leer ist.

Testen Sie Ihre Lösung an einigen einfachen Quelltexten, z. B. den Codebeispielen aus Kapitel 1 in JAPR01.

15 Pkt.

- b) Klammern können in einem Quelltext auch als Character (z. B. ' (') oder in einem String vorkommen. Diese Klammern müssen für eine robustere Lösung von der Prüfung auf korrekte Klammerung eines Quelltexts ausgeschlossen werden (Klammern in Kommentaren müssen Sie nicht berücksichtigen).

Ergänzen Sie Ihr Programm aus Aufgabenteil a), sodass Sie auch den Quelltext Ihres Programms überprüfen können:

```
java Klammerpruefung Klammerpruefung.java
```

Hinweis: Prüfen Sie, ob das aktuelle Zeichen ein einfaches oder doppeltes Anführungszeichen ist (beim Vergleich auf einfaches oder doppeltes Anführungszeichen müssen Sie eine Escape-Sequenz verwenden, z. B. '\ ' '). Wenn ja, ignorieren Sie alle Klammern, bis Sie wieder auf ein einfaches oder doppeltes Anführungszeichen stoßen.

5 Pkt.

Σ 20 Pkt.

4. Koch-Kurve (Java)

Die Koch-Kurve² ist eine fraktale Struktur, die ausgehend von einem Polygon mit den n Eckpunkten $\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_{n-1}$ jedes durch die beiden Punkte \mathbf{p}_i und $\mathbf{p}_{(i+1) \bmod n}$ definierte Linienstück („mod“ meint den Modulo-Operator) in vier kürzere Linienstücke unterteilt. Jedes entstehende Linienstück besitzt $\frac{1}{3}$ der Länge des ursprünglichen Linienstücks. Die Abb. 4.1 zeigt ein Beispielpolygon sowie verschiedene Unterteilungsstufen der entstehenden Koch-Kurven. In dieser Aufgabe sollen Sie schrittweise eine Visualisierung von Koch-Kurven in Java implementieren.

- a) Implementieren Sie eine von `JFrame` abgeleitete Klasse mit einem inneren `JPanel`. Die Größe des Frame bzw. Panel können Sie beliebig wählen.

Zeichnen Sie in das Fenster sodann ein Polygon.

Hinweise: Sie können zur Lösung der Aufgabe die Klasse `java.awt.Polygon` verwenden.³

Die Eckpunkte des Ausgangspolygons können Sie beliebig wählen. Im Beispiel verwenden wir ein Dreieck mit den Punkten

$$\mathbf{p}_0 = (250, 50), \quad \mathbf{p}_1 = (400, 350), \quad \mathbf{p}_2 = (100, 350)$$

10 Pkt.

- b) Implementieren Sie eine statische Methode

```
static Polygon subdivide(Polygon polygon)
```

die *einen* Unterteilungsschritt ausführt (das heißt, `subdivide` liefert für das Polygon in Abb. 4.1a das Polygon in Abb. 4.1b zurück).

Betrachten Sie hierzu die aus Wikipedia übernommene Abb. für die Unterteilung eines Linienstücks AE (Abb. 4.2): Die x -Koordinate des Punkts B in der Abb. ergibt sich durch eine Linearkombination der x -Koordinaten der beiden Punkte A und E :

$$B_x = \frac{2}{3}A_x + \frac{1}{3}E_x$$

Analog für die y -Koordinate:

$$B_y = \frac{2}{3}A_y + \frac{1}{3}E_y$$

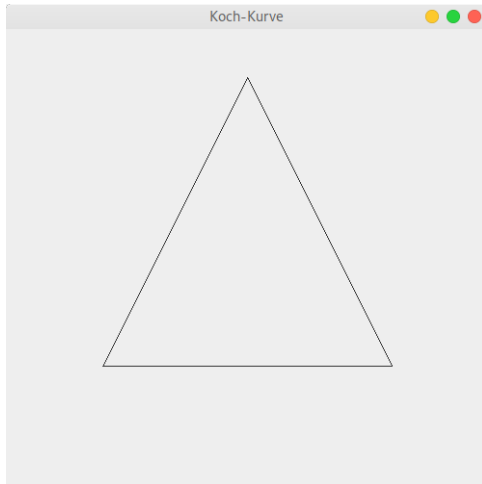
Für den Punkt D verwenden Sie die Faktoren $\frac{1}{3}$ und $\frac{2}{3}$.

² siehe <https://de.wikipedia.org/wiki/Koch-Kurve>

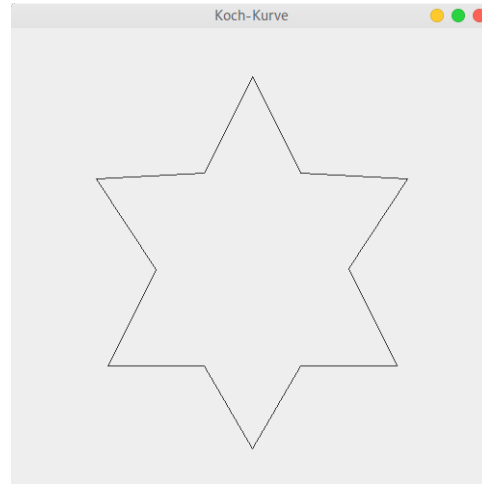
³ <https://docs.oracle.com/javase/10/docs/api/java/awt/Polygon.html>

Zur Berechnung des Punkts C können Sie wie folgt verfahren: Bestimmen Sie den Mittelpunkt \mathbf{p}_M der Linie AE . Berechnen Sie dann den Vektor

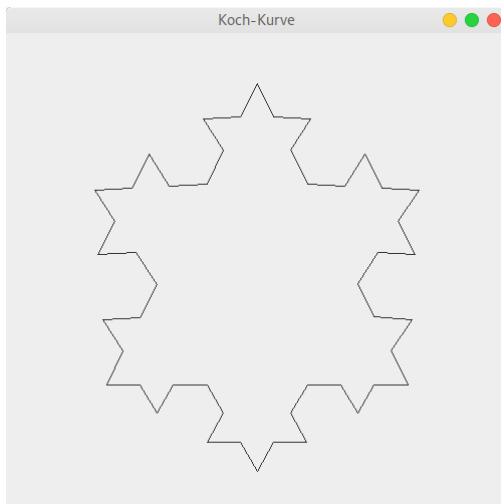
$$(x, y) = (E_x - A_x, E_y - A_y)$$



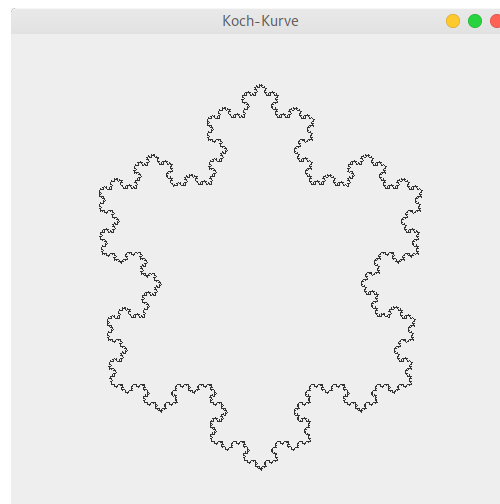
(a) Ausgangspolygon



(b) Koch-Kurve der Stufe 1



(c) Koch-Kurve der Stufe 2



(d) Koch-Kurve der Stufe 5

Abb. 4.1: Ausgangspolygon und Koch-Kurven der Stufen 1, 2 und 5

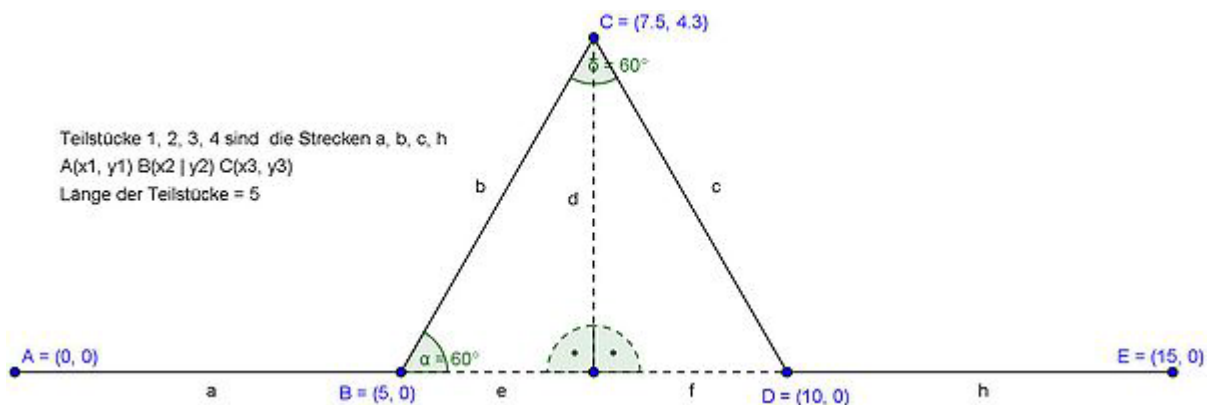


Abb. 4.2: Unterteilung des Liniensegments AB

Sie erhalten den zu der Linie AE *senkrechten* Vektor, indem Sie die x - und y -Koordinate vertauschen und das Vorzeichen von x negieren: $(y, -x)$. Diesen Vektor müssen Sie normieren, das heißt auf die Länge 1 bringen. Die Länge ℓ eines Vektors mit Koordinaten (x, y) bestimmt sich durch

$$\ell = \sqrt{x^2 + y^2}$$

Teilen Sie den zu AE senkrechten Vektor komponentenweise durch seine Länge. Nennen wir diesen normierten Vektor \mathbf{n} (\mathbf{n} zeigt in Richtung d in Abb. 4.2).

Die Längen von b und e sind bekannt ($\frac{1}{3}$ der Länge von AE bzw. $\frac{1}{6}$ der Länge von AE). Bestimmen Sie anhand des Satzes von Pythagoras die Länge von d (es gilt $b^2 = e^2 + d^2$).

Den Punkt C in Abb. 4.2 erhalten Sie, indem Sie den (normierten!) Vektor \mathbf{n} , skaliert mit der Länge von d , komponentenweise auf den Mittelpunkt \mathbf{p}_M addieren.

Gehen Sie nun für jedes Eckpunktepaar $\mathbf{p}_i, \mathbf{p}_{(i+1) \bmod n}$ des Polygons wie folgt vor:

- übernehmen Sie den Punkt \mathbf{p}_i in das neue Polygon
- übernehmen Sie den Punkt B aus Abb. 4.2 in das neue Polygon
- übernehmen Sie den Punkt C aus Abb. 4.2 in das neue Polygon
- übernehmen Sie den Punkt D aus Abb. 4.2 in das neue Polygon

Sie benötigen die Polygon-Eigenschaften `xpoints`, `ypoints` und `npoints` sowie die Polygon-Methode `addPoint`. Beachten Sie, dass `addPoint` *ganzzahlige* Koordinaten erwartet; Sie müssen die Fließkommawerte der berechneten Punkte somit in `int` casten. Bei der Berechnung der neuen Punkte B , C und D müssen Sie umgekehrt die ganzzahligen Polygon-Koordinaten in Fließkommawerte wandeln.

Wenn es Ihnen hilft, können Sie die folgende Klasse `Vector2` verwenden und ggf. weitere Methoden in dieser Klasse ergänzen:

```
public class Vector2 {
    public final double x;
    public final double y;
    public Vector2(double x, double y) {
        this.x = x; this.y = y;
    }
    public double length() {
        return Math.sqrt(x*x + y*y);
    }
    public Vector2 normalized() {
        double len = length();
        return new Vector2(x/len, y/len);
    }
    public Vector2 linearInterpolation(Vector2 other, double t) {
        return new Vector2((1-t)*x + t*other.x, (1-t)*y + t*other.y);
    }
}
```

15 Pkt.

c) Implementieren Sie nun eine Methode

```
static Polygon subdivide(Polygon polygon, int steps)
```

die das übergebene Polygon mittels der Methode `subdivide` aus Aufgabenteil b) rekursiv oder iterativ unterteilt, bis `steps` gleich 0 ist. Wenn Sie diese Methode beispielsweise für das Polygon aus Abb. 4.1a und `steps = 2` aufrufen, erhalten Sie das Ergebnis aus Abb. 4.1c.

Implementieren Sie zudem im Hauptprogramm (`main`) ein Auslesen der Aufrufparameter (`args`-Array), so dass die Unterteilungsstufe beim Aufruf des Programms angegeben werden kann:

```
java KochKurve 5
```

Falls kein Parameter übergeben wurde, oder dieser sich nicht in eine ganze Zahl wandeln lässt, verwenden Sie einen beliebigen Standardwert (z. B. 4).

Beachten Sie, dass aufgrund von Rundungsfehlern eine höhere Unterteilung als 5 nicht darstellbar ist.

5 Pkt.

Σ 30 Pkt.

Gesamt: 100 Pkt.