Aufgabenblatt 1

Kompetenzstufe 1

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Mittwoch, 30.10.2024 23:55 Uhr** in TUWEL hoch.
- Zusätzlich müssen Sie in TUWEL ankreuzen, welche Aufgaben Sie gelöst haben.
- Ihre Programme müssen kompilierbar und ausführbar sein.
- Ändern Sie bitte **nicht** die **Dateinamen** und die **vorhandene Ordnerstruktur**.
- Verwenden Sie, falls nicht anders angegeben, für alle Ausgaben `System.out.println()` bzw. `System.out.print()`.
- Verwenden Sie für die Lösung der Aufgaben keine Aufrufe (Klassen) aus der Java-API, außer diese sind ausdrücklich erlaubt.
- Erlaubt sind die Klassen `CodeDraw`, `Math` und `String` oder Klassen, die in den Hinweisen zu den einzelnen Aufgaben aufscheinen.
- Die Verwendung von Arrays ist nicht zulässig.

In diesem Aufgabenblatt werden folgende Themen behandelt:

- Verzweigungen (if-Anweisung, switch-Anweisung)
- Einfache Schleifen (for-Schleife, while-Schleife)
- Umgang mit den Klassen `String` und `CodeDraw`

Aufgabe 1 (1 Punkt)

Erweitern Sie die Methode `main`:

a) Schreiben Sie eine for-Schleife, die alle durch 21 teilbaren Zahlen im Intervall[1] [21, 420[aufsummiert und das Ergebnis auf der Konsole ausgibt.
Erwartetes Ergebnis: 3990

b) Schreiben Sie eine for-Schleife, die alle Zahlen, die durch 5 und nicht durch 9 teilbar sind, im Intervall von [45, 135] hintereinander und getrennt durch Doppelpunkt (`':'`) ausgibt. Zusätzlich wird noch vor der ersten Zahl und nach der letzten Zahl ein Doppelpunkt ausgegeben.
Erwartetes Ergebnis: :50:55:60:65:70:75:80:85:95:100:105:110:115:120:125:130:

c) Schreiben Sie eine for-Schleife, die alle Zeichen der ASCII[2]-Werte im Intervall]50, 71[in absteigender Reihenfolge durch Beistriche getrennt ausgibt.
Erwartetes Ergebnis: F,E,D,C,B,A,@,?,>,=,<,;,:,9,8,7,6,5,4,3

d) Schreiben Sie eine for-Schleife, die alle Vorkommen der Buchstaben `'z'` und `'Z'` im Satz `Zehn zahme Ziegen zogen ziemlich zügig zehn Zentner Zucker zum Zoo!` zählt und das Ergebnis auf der Konsole ausgibt.
Erwartetes Ergebnis: 11

[1] Intervall-Schreibweise: `https://de.wikipedia.org/wiki/Intervall_(Mathematik)`
[2] ASCII-Code: `https://de.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange`

Aufgabe 2 (1 Punkt)

Erweitern Sie die Methode `main`:

- Deklarieren Sie eine String-Variable `text` und initialisieren Sie diese mit `"Eine nennenswerte und geeignete Sprache."`. Testen Sie zusätzlich mit dem String `"Anzahl der Zeichen ist nicht genug!"`, um Ihre Implementierung zu prüfen. Sie dürfen auch mehrere String-Variablen deklarieren und durch entsprechende Zuweisungen unterschiedliche Tests realisieren.

a) Schreiben Sie eine while-Schleife, die vom String `text` von vorne beginnend jedes Zeichen ausgibt. Zusätzlich wird überprüft, ob es sich bei dem Zeichen um einen Selbstlaut (Vokal) handelt. Ist das Zeichen ein Selbstlaut, dann wird vor diesem Zeichen noch das Zeichen `'-'` ausgegeben.
`"Eine nennenswerte und geeignete Sprache."` **liefert**
`"-E-in-e n-enn-ensw-ert-e -und g-e-e-ign-et-e Spr-ach-e."`
`"Anzahl der Zeichen ist nicht genug!"` **liefert**
`"-Anz-ahl d-er Z-e-ich-en -ist n-icht g-en-ug!"`

b) Schreiben Sie eine while-Schleife, die im String `text` alle Zeichen die nicht Buchstaben im Alphabet zwischen `'a'` bis `'j'` (beides inklusive und auch als Großbuchstaben) und die keine Leerzeichen (`' '`) sind kopiert und in einem neuen String speichert. Anschließend wird der neue String auf der Konsole ausgegeben.
Erwartetes Ergebnis:
`"Eine nennenswerte und geeignete Sprache."` **liefert** `"nnnnnswrtunntSpr."`
`"Anzahl der Zeichen ist nicht genug!"` **liefert** `"nzlrZnstntnu!"`

c) Schreiben Sie eine while-Schleife, die alle durch 5 und durch 7 teilbaren Zahlen im Intervall [35, 175[nebeneinander durch Leerzeichen getrennt ausgibt.
Erwartetes Ergebnis: 35 70 105 140

Aufgabe 3 (2 Punkte)

Realisieren Sie folgende Aufgabenstellung und erweitern Sie dazu die Methode `main`:

Bei dieser Aufgabe soll die Simulation eines nicht linearen Systems nachgebildet werden. Dazu wird ein Beispiel aus einem Buch über Systemdynamik[3] übernommen und abgeändert. Bitte lesen Sie sich einmal den Text genau durch. Der Text beschreibt den Hintergrund für die im Anschluss angeführten Formeln. Für das Programm müssen Sie dann entscheiden, welche Variablen Sie anlegen und wie Sie diese initialisieren (achten Sie auf die Hinweise im Text). Danach müssen Sie die unten angegebenen Berechnungen mehrmals (500 mal) ausführen und in jeder Iteration Informationen auf der Kommandozeile ausgeben.

Beschreibung: Es soll die Beziehung zwischen einer Population von Hasen (Beute) und Füchsen (Räuber) in einem abgesteckten Weidegebiet nachgebildet werden. Gibt es keine Füchse, würde die Hasenpopulation bis zu dem Punkt anwachsen, an dem die Tragfähigkeit (Kapazität) des abgesteckten Weidegebiets ihre Grenzen erreicht. Gibt es keine Hasen, dann haben die Füchse keine Nahrung und würden verhungern. Wenn Hasen und Füchse zusammen leben, dann haben Füchse Nahrung, das heißt, es wird der Hasenpopulation Energie entzogen und der Fuchspopulation zugeführt. Durch Abnahme der Hasen wird wieder Kapazität des Weidegebiets frei, wodurch die Hasenpopulation wieder anwachsen kann. Ziel soll es sein, diesen Zusammenhang in Code abzubilden und die Dynamik (Schwingungsvorgänge) dieses nicht linearen Systems bei gegebenen Anfangsgrößen zu beobachten und die Ergebnisse auf der Kommandozeile auszugeben. Für die Durchführung müssen einige Annahmen getroffen werden.

- Die freie Kapazität (`freeCap`) des Weidegebiets ergibt sich aus der Differenz der maximalen Tragfähigkeit (`maxCap`) der Weide und der aktuellen Hasenpopulation (`numRabbits`).
- Füchse (`numFoxes`) benötigen für ihre Lebensvorgänge (Respiration) Energie und es wird angenommen, dass die Füchse Respirationsverluste (`decFoxes`) von 20% ihrer Biomasse pro Woche haben.
- Es wird angenommen, dass die Hasen (`numRabbits`) bei optimalen Bedingungen (unbegrenzte Tragfähigkeit des Weidegebiets) eine Wachstumsrate von 8% pro Woche aufweisen. Abhängig von der Größe der Weide muss diese jedoch an die Größe der Hasenpopulation angepasst werden, und bis auf 0 zurückgehen, sobald man sich der maximalen Tragfähigkeit der Weide nähert. Daher muss für die Berechnung der Populationszunahme (`incRabbits`) die Wachstumsrate mit dem Verhältnis `freeCap/maxCap` multipliziert werden.
- Die Gewichtung der Verluste der Hasen und Gewinne der Füchse durch Reißen der Beute wird durch die Annahme bestimmt, dass die normale Verteilung im Weidegebiet bei 500 Hasen und 10 Füchse liegt. Das heißt, es ergeben sich bei 10 Füchsen Respirationsverluste von 2 Füchsen (20%) pro Woche, die es gilt auszugleichen. Die Multiplikation aus Hasen und Füchse ergibt ein Maß für die Treffen (`numContacts`) beider Tierarten. Im Normalzustand ergeben sich daraus 5000 Kontakte beider Tierarten. Daraus resultiert eine Gewichtung von 0.0004 (2/5000) für die Berechnung der neuen Fuchspopulation. Bei der Annahme, dass

[3] Hartmut Bossel, *Systemdynamik: Grundwissen, Methoden und BASIC-Programme zur Simulation dynamischer Systeme*, Vieweg+Teubner Verlag, 1987

eine Fuchseinheit 5 Hasen entspricht, ergibt sich eine Gewichtung für die Berechnung der Hasenpopulation von 0.002 (2*5/5000).

- Für die Simulation wird noch ein Zeitintervall `dt` benötigt, das hier mit 1 angenommen wird und fiktiv einer Woche entsprechen soll. Die Variable `dt` bleibt während der Simulation über konstant und ist hier nicht als Laufvariable für die Schleife zu verwenden.

Mit den oben stehenden Annahmen können folgende Gleichungen aufgestellt werden, die den nicht linearen Zusammenhang von Füchsen und Hasen darstellen:

```
freeCap = maxCap - numRabbits;
incRabbits = (1.0 / maxCap) * freeCap * 0.08 * numRabbits;
decFoxes = 0.2 * numFoxes;
numContacts = numRabbits * numFoxes;
numRabbits = numRabbits + dt * (incRabbits - 0.002 * numContacts);
numFoxes = numFoxes + dt * (0.0004 * numContacts - decFoxes);
```

- Führen Sie eine konkrete Simulation mit den folgenden Startwerten durch: Maximale Tragfähigkeit des Weidegebiets `maxCap = 1500`, Anzahl der Füchse `numFoxes = 2` und Anzahl der Hasen `numRabbits = 2500`. Lassen Sie die Simulation 500 Iterationen laufen. Beachten Sie, dass die Simulation mit Gleitkommaarithmetik arbeitet und es daher nicht nur ganze Tiereinheiten gibt.

- Zusätzlich soll eine Konsolenausgabe generiert werden, die die Werte in den einzelnen Iterationsschritten ausgibt und die Anzahl der Hasen und Füchse mittels Symbole als Balken darstellt. Für die numerische Ausgabe verwenden Sie die Methode `System.out.format(...)`[4], bei welcher Sie vorhandene Gleitkommazahlen mit einer Genauigkeit von zwei Stellen ausgeben. In den Balken werden durch den Character `'*'` die Hasen repräsentiert und durch den Character `'#'` die Füchse. Die Balken haben eine Maximalbreite von 500 Zeichen. Gehen Sie dabei von einer maximalen Anzahl von 2500 Hasen und 200 Füchsen aus, d.h. 2500 Hasen würden durch 500 `'*'`-Zeichen und 200 Füchse durch 500 `'#'`-Zeichen repräsentiert werden. Das wiederum bedeutet z.B. dass 5 Hasen durch einen `'*'` repräsentiert werden und 2 Füchse durch 5 `'#'`-Zeichen. Bitte beachten Sie aber, dass Sie diese Verhältnisse selbst berechnen müssen und dass nur ganze Zeichen dargestellt werden können (z.B. entspricht 1 Fuchs 2,5 Zeichen und daher muss vor der Darstellung z.B. mit `Math.round` auf 3 aufgerundet werden).

  Zum Beispiel sieht die Ausgabe für die Iterationsschritte 39 und 40 (bei den oben gegebenen Anfangswerten) dann in etwa so aus:

```
Iteration: 39 freeCap: 1367,08 numRabbits: 141,99 numFoxes: 1,98
**************************
#####

Iteration: 40 freeCap: 1358,01 numRabbits: 151,71 numFoxes: 1,70
***************************
####
```

[4] `https://docs.oracle.com/javase/tutorial/java/data/numberformat.html`

Am Ende der Simulation sehen die Werte in unserer Implementierung so aus:

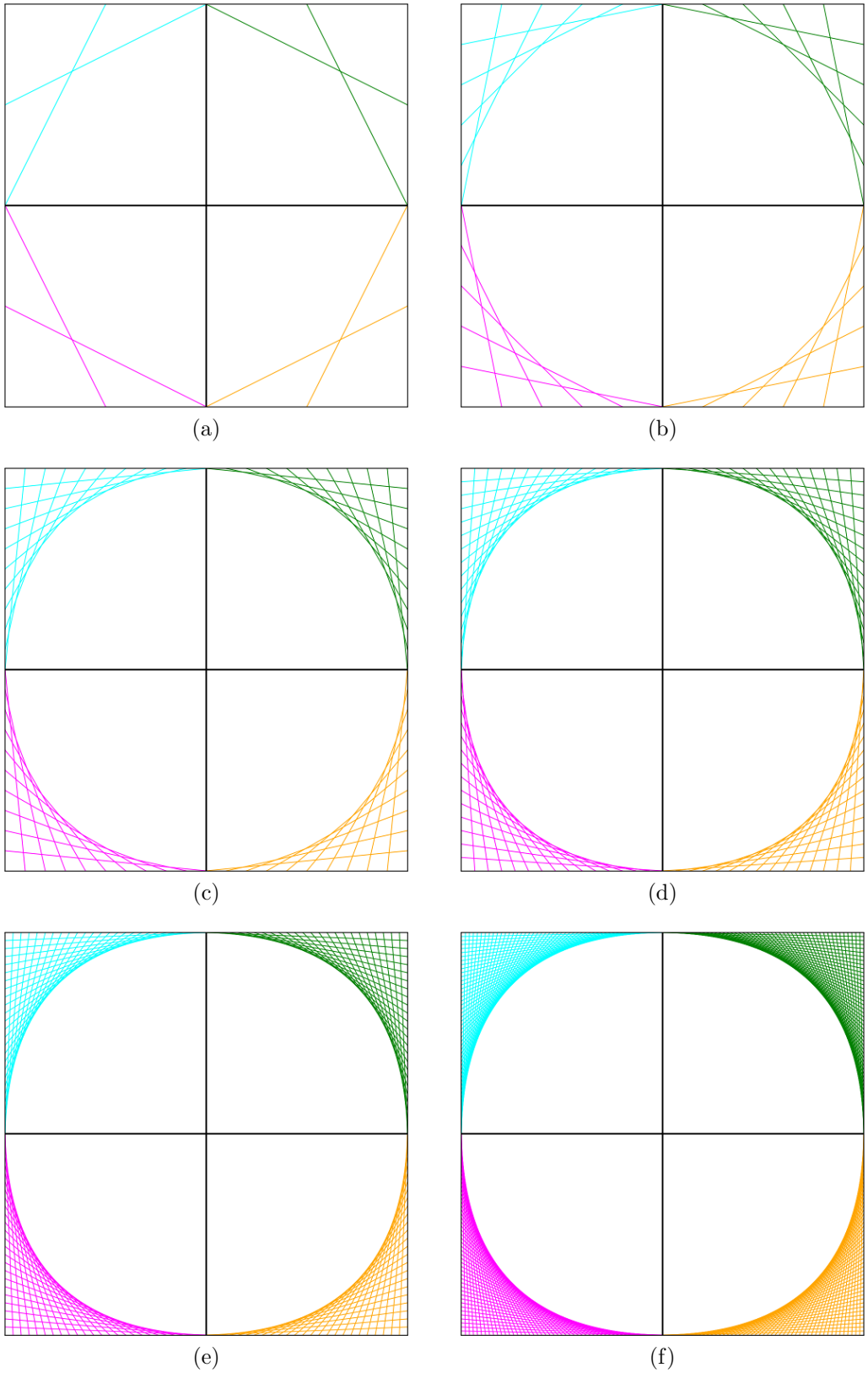
```
Iteration: 495 freeCap: 1016,34 numRabbits: 484,53 numFoxes: 26,03
**********************************************************************************************
############################################################

Iteration: 496 freeCap: 1015,47 numRabbits: 485,55 numFoxes: 25,87
**********************************************************************************************
############################################################

Iteration: 497 freeCap: 1014,45 numRabbits: 486,70 numFoxes: 25,72
**********************************************************************************************
############################################################

Iteration: 498 freeCap: 1013,30 numRabbits: 487,97 numFoxes: 25,58
**********************************************************************************************
############################################################

Iteration: 499 freeCap: 1012,03 numRabbits: 489,35 numFoxes: 25,46
**********************************************************************************************
############################################################

Iteration: 500 freeCap: 1010,65 numRabbits: 490,81 numFoxes: 25,35
**********************************************************************************************
############################################################
```

Aufgabe 4 (1 Punkt)

Erweitern Sie die Methode `main`:
Vervollständigen Sie das gegebene Programm, damit die in den Abbildungen 1a-1f gezeigten Bilder erzeugt werden. Im Moment werden die cyanfarbenen Linien im linken oberen Quadranten gezeichnet. Erweitern Sie das Programm an den vorgegebenen Stellen, damit die restlichen Linien (Muster) der drei anderen Quadranten gezeichnet werden.

Hinweise zum gegebenen Programm:

- Das Bild hat eine Größe von 500×500 Pixel und der Punkt $P(x = 0, y = 0)$ befindet sich in der oberen linken Ecke. Es wird ein Objekt `myDrawObj` der Klasse `CodeDraw` angelegt, um das Zeichenfenster mit der Größe 500×500 Pixel zu erstellen.
- Es existiert eine Variable `numLinesQuadrant` vom Typ `int`, die zur Einstellung dient, wie viele Linien pro Quadrant gezeichnet werden sollen. Die Abstände zwischen den Endpunkten der Linien auf den jeweiligen Achsen lassen sich aus der Anzahl der Linien bestimmen. Die Variable `numLinesQuadrant` wird direkt bei der Deklaration initialisiert.
- Mit `myDrawObj.setColor(Palette.CYAN)` wird die Zeichenfarbe auf `CYAN` gesetzt. Andere Farben in der Abbildung sind: `GREEN`, `MAGENTA` und `ORANGE`.
- Um die Zeichnung im Ausgabefenster sichtbar zu machen, wird die Methode `myDrawObj.show()` aufgerufen.
- Die Dokumentation unter dem Link `https://krassnig.github.io/CodeDrawJavaDoc/v5.0.x/codedraw/CodeDraw.html` kann Ihnen beim Verständnis der CodeDraw-Bibliothek helfen.



(a) (b) (c) (d) (e) (f)

Abbildung 1: Verschiedene Muster mit unterschiedlicher Anzahl an Linien `numLinesQuadrant`. a) `numLinesQuadrant = 2`, b) `numLinesQuadrant = 5`, c) `numLinesQuadrant = 10`, d) `numLinesQuadrant = 15`, e) `numLinesQuadrant = 25` und f) `numLinesQuadrant = 50`.

Aufgabe 5 (1 Punkt)

Erweitern Sie die Methode `main`:

- Erstellen Sie ein Ausgabefenster mit der Größe 300×300 Pixel.
- Schreiben Sie ein Programm, das unter Vorgabe einer Variable `n` die Abbildungen 2a-2f erzeugen kann. Die Variable `n` gibt an, wie viele Quadrate maximal horizontal, vertikal und diagonal im CodeDraw-Fenster Platz haben sollen und diese darf dabei nur ungerade Werte größer gleich 5 und kleiner gleich 19 annehmen. Überprüfen Sie in Ihrer Implementierung, ob die Vorgaben für `n` eingehalten werden, ansonsten geben Sie einen Hinweis (Fehlermeldung) auf der Konsole aus. Die blauen Quadrate haben eine Liniendicke von 2 Pixel. Die roten Quadrate haben eine Liniendicke von 4 Pixel. Achten Sie auf die Reihenfolge beim Zeichnen, da manchmal die blauen Kanten über den roten gezeichnet werden und manchmal die roten Kanten über den blauen Kanten gezeichnet werden. Dadurch kommt eine partielle Überlappung von Quadraten zustande.

(!) Hinweis: Verwenden Sie Gleitkommaarithmetik für die Berechnung der Abstände, um für jedes `n` das CodeDraw-Fenster füllend auszunutzen. Die Variable `n` wird direkt bei der Deklaration initialisiert und es ist kein Input von der Kommandozeile notwendig.

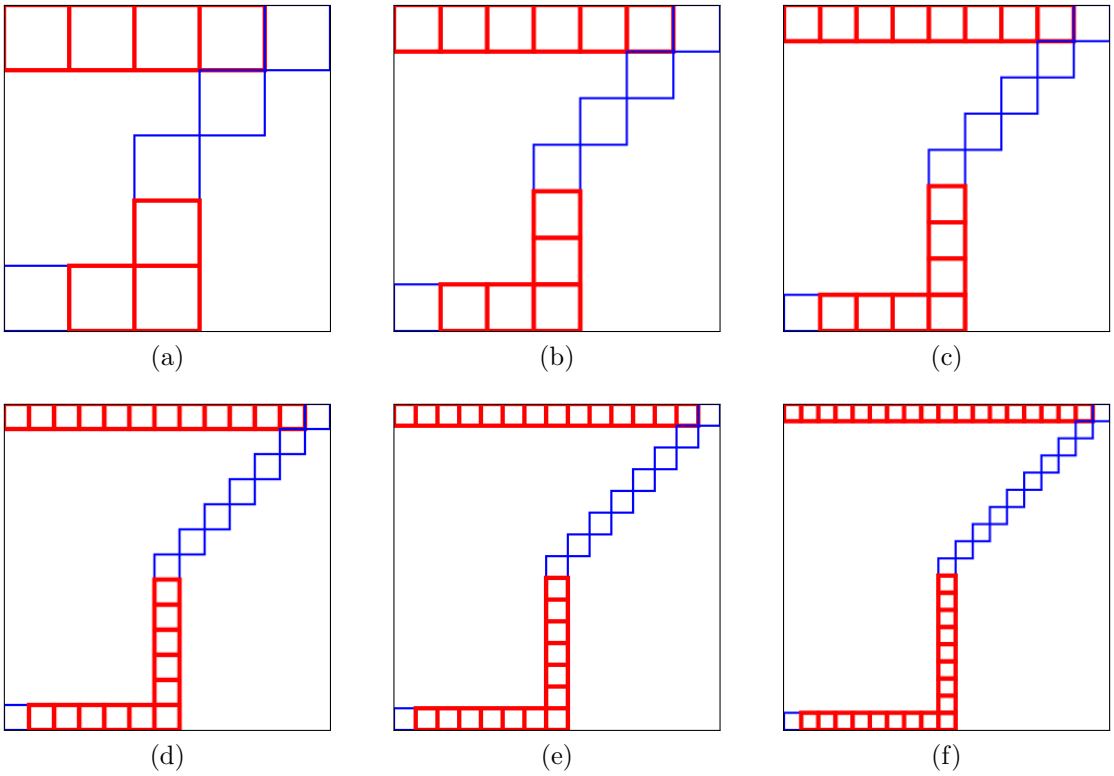


Abbildung 2: Verschiedene Quadratmuster mit unterschiedlicher Anzahl an Quadraten `n`. a) `n = 5`, b) `n = 7`, c) `n = 9`, d) `n = 13`, e) `n = 15` und f) `n = 19`.