Aufgabenblatt 3

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Mittwoch, 19.11.2025 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 `String`, `Math` und `CodeDraw`, es sei denn in den Hinweisen zu den einzelnen Aufgaben ist etwas anderes angegeben.
- Bitte beachten Sie die Vorbedingungen! Sie dürfen sich darauf verlassen, dass alle Aufrufe die genannten Vorbedingungen erfüllen. Sie müssen diese nicht in den Methoden überprüfen.

In diesem Aufgabenblatt werden folgende Themen behandelt:

- Implementieren von Methoden
- Überladen von Methoden
- Rekursion
- Rekursion und CodeDraw
- Vergleich von rekursiver und iterativer Implementierung

Aufgabe 1 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

- Implementieren Sie eine Methode `addChar`:

  ```
  void addChar(String text, char character)
  ```

  Diese Methode gibt einen String `text` formatiert aus. Es wird zwischen zwei Zeichen von `text` das Zeichen `character` abwechselnd zweimal bzw. einmal eingefügt. Geben Sie den veränderten String in einer Zeile auf der Konsole aus.
  Vorbedingung: `text != null`.

  Beispiel(e):
  `addChar("", '&')` liefert keine Ausgabe
  `addChar("A", '+')` liefert `A`
  `addChar("CW", '*')` liefert `C**W`
  `addChar("EP1", '-')` liefert `E--P-1`
  `addChar("Index", '#')` liefert `I##n#d##e#x`

- Implementieren Sie eine Methode `addChar`:

  ```
  void addChar(int number, char character)
  ```

  Diese Methode gibt eine Zahl `number` formatiert aus. Es wird zwischen zwei Ziffern von `number` das Zeichen `character` abwechselnd zweimal bzw. einmal eingefügt. Der resultierende String wird in einer Zeile auf der Konsole ausgegeben. Für die Realisierung der Methode darf die Zahl in einen String umgewandelt werden (`Integer.toString(...)`). Vermeiden Sie redundanten Code, indem Sie eine bereits implementierte Methode verwenden.
  Vorbedingung: `number` $\geq$ `0`.

  Beispiel(e):
  `addChar(1, '.')` liefert `1`
  `addChar(42, ':')` liefert `4::2`
  `addChar(148, '$')` liefert `1$$4$8`
  `addChar(2048, ')')` liefert `2))0)4))8`
  `addChar(131719, '%')` liefert `1%%3%1%%7%1%%9`

- Implementieren Sie eine Methode `addChar`:

  ```
  void addChar(String text, String characters)
  ```

  Diese Methode gibt einen String `text` unterschiedlich formatiert aus. Erst wird zwischen zwei Zeichen von `text` das erste Zeichen von `characters` abwechselnd zweimal bzw. einmal eingefügt und auf der Konsole in einer Zeile ausgegeben. Danach wird das zweite Zeichen von `characters` abwechselnd zweimal bzw. einmal zwischen den Zeichen von `text` eingefügt und auf der Konsole in einer weiteren Zeile ausgegeben, usw. Dies soll für jedes Zeichen aus dem String `characters` geschehen, d.h. der String `text` wird mit verschiedenen Zeichen formatiert mehrmals ausgegeben. Vermeiden Sie redundanten Code, indem Sie eine bereits implementierte Methode verwenden.
  Vorbedingungen: `text != null` und `characters != null`.

  Beispiel(e):
  `addChar("CW", "!O(")` liefert
  `C!!W`
  `COOW`
  `C((W`
  `addChar("Index", "T1#+")` liefert
  `ITTnTdTTeTx`
  `I11n1d11e1x`
  `I##n#d##e#x`
  `I++n+d++e+x`

- Implementieren Sie eine Methode `addChar`:

  ```
  void addChar(String text)
  ```

  Diese Methode gibt einen String `text` formatiert aus. Es wird zwischen zwei Zeichen von `text` das Zeichen `=` abwechselnd zweimal bzw. einmal eingefügt und in einer Zeile auf der Konsole ausgegeben. Vermeiden Sie redundanten Code, indem Sie eine bereits implementierte Methode verwenden.
  Vorbedingung: `text != null`.

  Beispiel(e):
  `addChar("")` liefert keine Ausgabe
  `addChar("CW")` liefert `C==W`
  `addChar("EP1")` liefert `E==P=1`

Aufgabe 2 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

(!) Gilt für alle zu implementierenden Methoden: Sie dürfen keine Klassenvariablen oder zusätzliche eigene Hilfsmethoden verwenden. Die vorgegebenen Methodenköpfe dürfen nicht erweitert oder geändert werden. Für die Implementierung der Aufgabenstellung dürfen **keine Schleifen** verwendet werden. Sie dürfen für diese Aufgabe nur die Methoden `length()`, `isEmpty()`, `charAt()` und `substring()` der String-Klasse verwenden.

- Implementieren Sie eine **rekursive** Methode `printEvenNumbersDescending`:

  ```
  void printEvenNumbersDescending(int end)
  ```

  Diese Methode gibt alle geraden Zahlen im Intervall von [0, `end`] **absteigend** in der Konsole aus. Geben Sie die einzelnen Werte getrennt durch Leerzeichen nebeneinander aus. Es darf auch nach der letzten Zahl ein Leerzeichen ausgegeben werden.
  Vorbedingung: `end` ≥ 0.

  Beispiel:
  `printEvenNumbersDescending(14)` liefert 14 12 10 8 6 4 2 0

- Implementieren Sie eine **rekursive** Methode `countEqualNeighbors`:

  ```
  int countEqualNeighbors(String text)
  ```

  Diese Methode zählt, wie oft zwei direkt hintereinander vorkommende Zeichen im String `text` gleich sind, und gibt diese Anzahl zurück. Zum Beispiel hat der String `"ABBBC"` zweimal gleiche Nachbarn, da die ersten zwei `B` und die zweiten zwei `B` gleiche Nachbarpaare bilden.
  Vorbedingung: `text != null`.

  Beispiele:
  `countEqualNeighbors("AA")` liefert 1
  `countEqualNeighbors("ABBBCCDDDE")` liefert 5
  `countEqualNeighbors("matt")` liefert 1
  `countEqualNeighbors("Schifffahrt")` liefert 2

Aufgabe 3 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

(!) Sie dürfen bei dieser Methode keine Klassenvariablen oder zusätzliche eigene Hilfsmethoden verwenden. Der vorgegebene Methodenkopf darf nicht erweitert oder geändert werden. Für die Implementierung der Aufgabenstellung dürfen **keine Schleifen oder Arrays** verwendet werden. Sie dürfen für diese Aufgabe nur die Methoden `length()`, `isEmpty()`, `charAt()` und `substring()` der String-Klasse verwenden.

- Implementieren Sie eine **rekursive** Methode `checkBrackets`:

  ```
  String checkBrackets(String text)
  ```

  Diese Methode gibt einen neuen String zurück, der nur den Substring innerhalb der äußersten öffnenden und schließenden Klammer von `text` enthält. Die Klammerzeichen `'['` und `']'` der äußersten geschlossenen Klammer werden dabei entfernt. Kommt in `text` kein `'['` vor einem `']'` vor, wird ein leerer String zurückgeliefert.
  Vorbedingung: `text != null`.

  Beispiele:
  `checkBrackets("HGT[CDE]HJH")` liefert `"CDE"`
  `checkBrackets("aa[[b]")` liefert `"[b"`
  `checkBrackets("abc")` liefert `""`
  `checkBrackets("[a")` liefert `""`
  `checkBrackets("]b[")` liefert `""`

Aufgabe 4 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

(!) Sie dürfen bei dieser Methode keine Klassenvariablen oder zusätzliche eigene Hilfsmethoden verwenden. Der vorgegebene Methodenkopf darf nicht erweitert oder geändert werden. Für die Implementierung der Aufgabenstellung dürfen **keine Schleifen oder Arrays** verwendet werden. Sie dürfen für diese Aufgabe nur die Methoden `length()`, `isEmpty()`, `charAt()` und `substring()` der String-Klasse verwenden.

- Implementieren Sie eine **rekursive** Methode `appendAllSignsLeft`:

  ```
  String appendAllSignsLeft(String text, char character)
  ```

  Diese Methode fügt alle Zeichen `character` die innerhalb des Strings `text` gefunden werden ganz vorne nochmals hinzu. Das Ergebnis wird als neuer String zurückgeliefert.
  Vorbedingungen: `text != null`.

  Beispiele:
  `appendAllSignsLeft("az3kj", 'z')` liefert `"zaz3kj"`
  `appendAllSignsLeft("kjdn{nd8xngs+d#k", 'n')` liefert `"nnnkjdn{nd8xngs+d#k"`
  `appendAllSignsLeft("", 'e')` liefert `""`
  `appendAllSignsLeft("4", '4')` liefert `"44"`
  `appendAllSignsLeft("ji)o3ie6pk(2i",'i')` liefert `"iiiji)o3ie6pk(2i"`
  `appendAllSignsLeft("nothing", 'x')` liefert `"nothing"`

Aufgabe 5 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

(!) Sie dürfen für die folgende rekursive Methode `drawRecursiveSquares` keine Klassenvariablen oder zusätzliche eigene Hilfsmethoden verwenden. Der vorgegebene Methodenkopf darf nicht erweitert oder geändert werden. Für die Implementierung der Methode `drawRecursiveSquares` darf **keine Schleife** verwendet werden.

- Implementieren Sie die **rekursive** Methode `drawRecursiveSquares`:

```
void drawRecursiveSquares(CodeDraw myDrawObj, int x, int y, int s)
```

Diese Methode zeichnet überlappende Quadrate. Die Methode hat die Parameter `x` und `y`, welche den Koordinaten der Quadratmittelpunkte entsprechen. Zusätzlich wird mit dem Parameter `s` die Seitenlänge der Quadrate angegeben. Mit diesen Parametern wird ein gefülltes Quadrat gezeichnet. Geben Sie acht, dass die größeren Quadrate die kleineren Quadrate überdecken und nicht umgekehrt. Die Füllfarbe der Quadrate ist `Color.YELLOW` und die Umrandung `Color.BLACK`.

Erzeugen Sie in `main` ein Fenster der Größe 512×512 Pixel und rufen Sie die Methode mit `drawRecursiveSquares(myDrawObj, 256, 256, 256)` auf. Der Aufruf erzeugt dann durch Selbstaufrufe der Methode `drawRecursiveSquares` ein Quadratmuster, wie in Abbildung 1 dargestellt. Bei jedem rekursiven Aufruf wird der Mittelpunkt des nächsten Quadrates um die Länge `s/2` in x- und y-Richtung verschoben (in jede der vier Diagonalrichtungen). Die Seitenlänge `s` des Quadrats halbiert sich bei jedem Rekursionsschritt. Die Rekursion wird solange fortgeführt, solange die Seitenlänge `s` $\geq$ `4` ist.

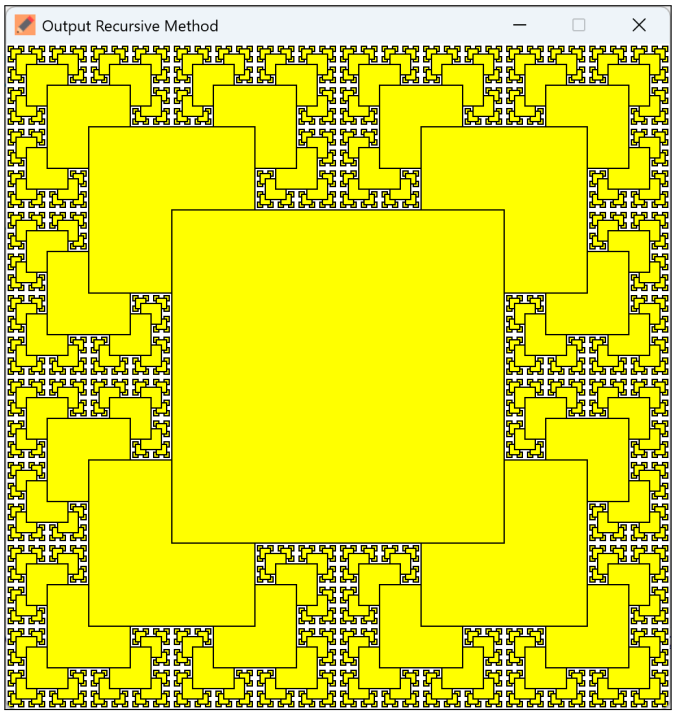


Abbildung 1: Rekursives Quadratmuster.

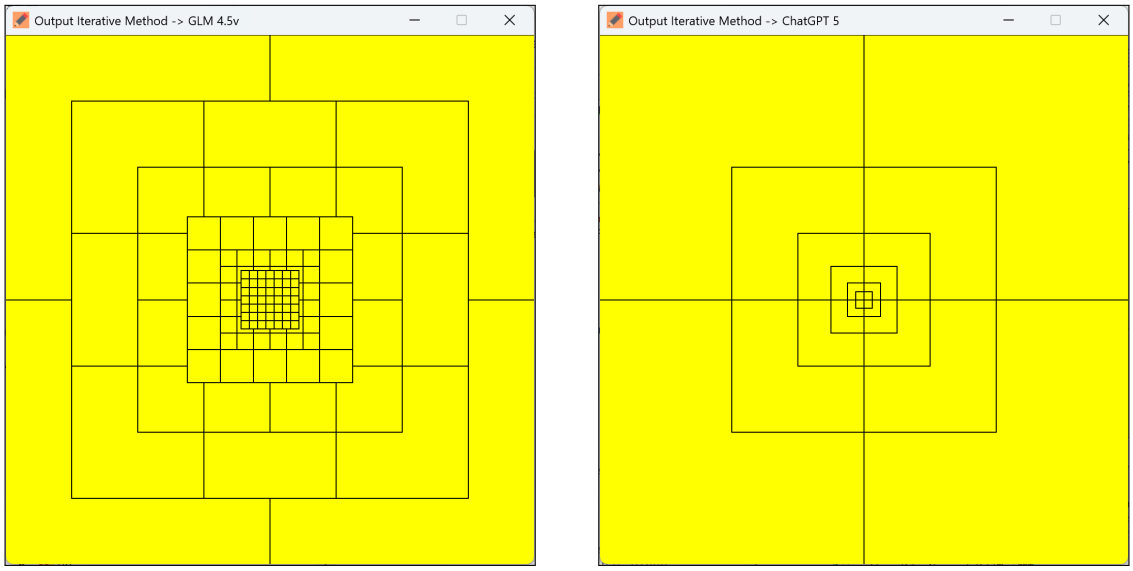
Aufgabe 6 (1 Punkt)

Implementieren Sie folgende Aufgabenstellung:

- Implementieren Sie die **iterative** Methode `drawIterativeSquares`:

  ```
  void drawIterativeSquares(CodeDraw myDrawObj, int width)
  ```

  Diese Methode soll das gleiche Ergebnis liefern wie die rekursive Methode in Aufgabe 5. Wir haben unter Verwendung von ChatGPT [1][2] (**G**enerative **P**re-trained **T**ransformer[3]) und Z.ai [4][5] eine iterative Variante erzeugen lassen. In Abbildung 2a ist das Ergebnis bei Verwendung von Z.ai Version GLM4.5v zu sehen und in Abbildung 2b das Ergebnis bei Verwendung der ChatGPT Version 5. Es wurde die Beschreibung der rekursiven Methode genommen und eine Anfrage bei *ChatGPT* und *Z.ai* gestellt, eine iterative Variante zu generieren. Zusätzlich wurde der Hinweis angegeben, dass die Implementierung in JAVA, unter der Zuhilfenahme der Bibliothek *CodeDraw*, erfolgen soll. Anschließend wurde noch ergänzt, dass die Methodenköpfe nicht verändert werden dürfen und auch die Verwendung von Arrays nicht erlaubt ist.



(a) (b)

Abbildung 2: Iteratives Quadratmuster a) mit *Z.ai* GLM4.5v und b) mit *ChatGPT* 5 generiert.

Analysieren Sie die von **Z.ai** erstellte Version (`drawIterativeSquaresGLM45v(...)`) und die von **ChatGPT** erstellte Version (`drawIterativeSquaresChatGPT5(...)`) genau. Überlegen Sie, was umgebaut werden muss, um eine korrekte Version zu erhalten. Schreiben Sie eine richtige und korrekte iterative Methode (`drawIterativeSquares()`), die das gezeigte Ergebnis in Abbildung 1 aus Aufgabe 5 liefert.

[1] `https://openai.com/blog/chatgpt`
[2] `https://en.wikipedia.org/wiki/ChatGPT`
[3] `https://en.wikipedia.org/wiki/Generative_pre-trained_transformer`
[4] `https://z.ai/chat`
[5] `https://en.wikipedia.org/wiki/Z.ai`