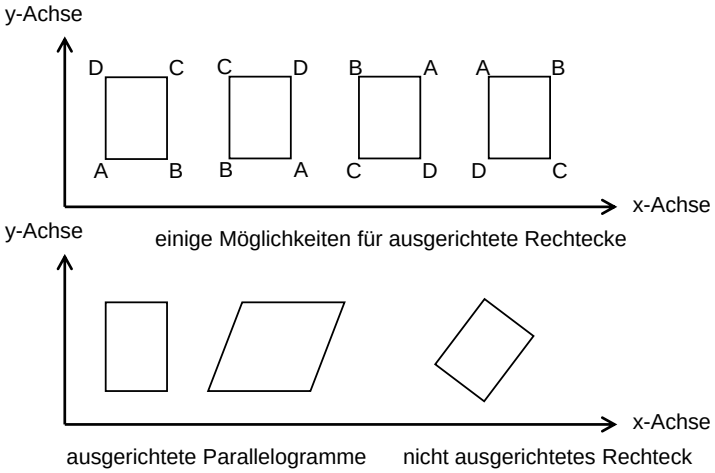
Prof. Dr. Stephan Kleuker
Hochschule Osnabrück
Fakultät Ing.-Wissenschaften und Informatik
- Software-Entwicklung -

Programmierung 1

Wintersemester 2021/22
Aufgabenblatt 11

36. Aufgabe (8 Punkte, Nutzung Arrays)

Gegeben sei das Projekt von der Webseite der Veranstaltung. Ergänzen Sie die Klasse Viereck, das aus vier beliebigen Punkten besteht, die in einem Array verwaltet werden. Viereck soll das gegebene Interface ViereckInterface realisieren. Gehen Sie davon aus, dass die Punkte in der Reihenfolge A, B, C, D, wie in der Abbildung gezeigt, eingegeben werden und die Kanten von A nach B, B nach C, C nach D und D nach A verlaufen, dabei muss A aber nicht die linke untere Ecke sein.



Sollte bei den Aufgaben c) bis g) einer der Punkte null oder das Viereck nicht OK sein, ist das Ergebnis false. Prüfen Sie Ihre Implementierungen mit den vorher zu lesenden JUnit-Tests.

a) Ergänzen Sie in der gegebenen Klasse Punkt eine Methode abstand(), die einen Punkt übergeben bekommt und den Abstand der Punkte als double-Wert zurückgibt. Sollte der übergebene Punkt null sein, ist das Ergebnis 0.0. Sie können Klassenmethoden wie Math.sqrt(.) nutzen und sich an Pythagoras erinnern.
b) Schreiben Sie einen Konstruktor für Viereck, der vier Punkte übergeben bekommt.
c) Schreiben Sie eine Methode istOK(), die genau dann true als Ergebnis liefert, wenn es sich um ein echtes Viereck handelt, also alle vier Punkte unterschiedlich sind und maximal zwei Punkte den gleichen x-Wert sowie maximal zwei Punkte den gleichen y-Wert haben. (Man kann sich überlegen, dass es trotzdem noch „hässliche“ Vierecke geben kann (z. B. (1,1), (2,2), (3,3), (4,4)), das ist hier zu ignorieren. Dies ist aber eine spannende, nicht triviale Zusatzaufgabe.)
d) Schreiben Sie eine Methode istAusgerichtet(), die genau dann true als Ergebnis liefert, wenn es sich um ein echtes „ausgerichtetes“ Viereck handelt. Dabei heißt ein Viereck hier ausgerichtet, wenn zwei nicht verbundene Kanten parallel zur x-Achse verlaufen. Oben sehen Sie 6 ausgerichtete Vierecke (rechts unten nicht).
e) Schreiben Sie eine Methode istAusgerichtetesParallelogramm(), die genau dann true als Ergebnis liefert, wenn es sich bei dem Viereck um ein echtes „ausgerichtetes“ (siehe d)) Parallelogramm handelt, also die parallelen Seiten gleich lang sind. Oben sehen Sie 6 ausgerichtete Parallelogramme (rechts unten nicht).
f) Schreiben Sie eine Methode istAusgerichtetesRechteck(), die genau dann true als Ergebnis liefert, wenn es sich bei dem Viereck um ein echtes „ausgerichtetes“ (siehe d)) Rechteck handelt. Also ein Rechteck parallel zur x-Achse und y-Achse. Oben sehen Sie 5 ausgerichtete Rechtecke.
g) Schreiben Sie eine Methode istRaute(), die genau dann true als Ergebnis liefert, wenn alle vier Kanten gleich lang sind. Erlauben Sie bei der Prüfung der Werte eine minimale Ungenauigkeit, um Probleme mit der Rechengenauigkeit zu vermeiden.
h) Schreiben Sie eine equals() Methode für Vierecke, dabei reicht es aus, wenn die vier Kanten übereinstimmen (auch null-Werte erlaubt) und es ist unerheblich ob ein Viereck OK ist. Es müssen also nur die vier Punkte in der gleichen Weise verbunden sein. Also ein Viereck mit punkten {p1,p2,null,p3} ist u. a.equals mit {null,p3,p1,p2}.

37. Aufgabe (10 Punkte, Nutzung mehrdimensionaler Arrays)

Zur Veranschaulichung des Ergebnisses der letzten Teilaufgabe kann von der Veranstaltungsseite das Programm gameOfLife.exe geladen und in Windows 10

ausgeführt werden. Da das Programm nicht in Java geschrieben ist, sieht das Interaktionsbrett etwas anders aus. Das Programm wird mit einem Klick auf „X“ rechts-oben beendet.

Schreiben Sie eine Simulation von Conways „Game of Life“. Die Simulation läuft auf einem Gitter der Größe n*n, wobei n vom Nutzer eingegeben werden kann. Die einzelnen Elemente werden Zellen genannt, die zwei Zustände haben, „bewohnt“ (schwarz dargestellt) oder „unbewohnt“ (weiß dargestellt). Mit jedem Simulationsschritt wird der Zustand jeder Zelle nach folgenden Regeln verändert: Jede bewohnte Zelle mit genau zwei oder genau drei bewohnten Nachbarn bleibt bewohnt (blauer Fall), sonst wird sie unbewohnt (grüner Fall). Jede unbewohnte Zelle mit genau drei bewohnten Nachbarn wird bewohnt (roter Fall), bleibt sonst unbewohnt (oranger Fall). Jede Zelle hat damit minimal drei (in der Ecke) und maximal acht Nachbarn (in der Mitte).

Probieren Sie zunächst auf Papier einige Beispiele aus, wie sich die Zellen verändern können. Gibt es z. B. Strukturen, die, solange sich ihr Umfeld nicht ändert, immer bewohnt bleiben?

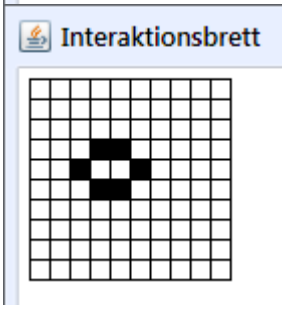
a) Nutzen Sie als Datenstruktur ein zweidimensionales Array. Schreiben Sie einen Nutzungsdialog, mit dem n und dann eine Zahl w für eine prozentuale Wahrscheinlichkeit zwischen 0 und 100 eingegeben wird.

b) Füllen Sie dann das Array zufällig mit bewohnten Zellen, dabei soll die Anzahl der bewohnten Zellen vom Wert w abhängen.

c) Schreiben Sie eine Möglichkeit, das Array mit Hilfe eines Interaktionsbrett-Objekts zu visualisieren. In den Abbildungen sind bewohne Zellen schwarz.

d) Implementieren Sie das beschriebene Verfahren, mit dem die bewohnten Zellen nach einem Schritt berechnet werden. Ergänzen Sie den Nutzungsdialog so, dass der Nutzer eine Schrittzahl angeben kann, dann diese Anzahl von Schritten ausgeführt und das Ergebnis angezeigt wird (ein Interaktionsbrett kann mit der Methode abwischen() gelöscht werden). Überlegen Sie sich eine sinnvolle Abbruchmöglichkeit, die Bilder am Rand zeigen einen typischen Ablauf, Eingaben sind umrandet.


```
Feldgroesse n (>0): 10
Wahrscheinlichkeit w (0-100): 30
Anzahl Schritte (>0): 1
Anzahl Schritte (>0): 1
Anzahl Schritte (>0): 1
Anzahl Schritte (>0): 50
Anzahl Schritte (>0): Ende
```



e) (freiwillig) Stellen Sie Ihr Programm so um, dass durch Anklicken der Felder den Zustand der Zellen verändert werden kann. Es könnte weiterhin einen Knopf geben, mit dem ein Schritt durchgeführt wird (nicht in Beispiellösung). Hier ist es sinnvoll, das Gitter mit echten Objekten nachzubilden, die die einzelnen Zellen verwalten.

Hinweis: Überlegen Sie sich als Ansatz, warum es sinnvoll sein kann mit zwei zweidimensionalen Arrays zu arbeiten.