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

**Programmierung 1**

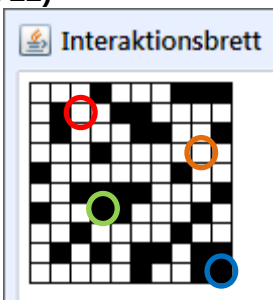
Wintersemester 2025/26
**Aufgabenblatt 12**

Nur zum Selbststudium!

### 0.12 Aufgabe

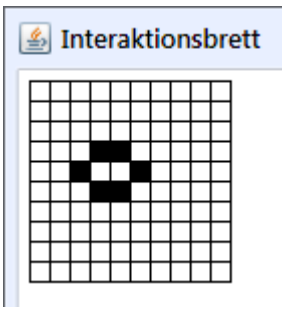
Geben Sie die Lösungsworte der Quizze aus der Lernnotiz an.

### 39. Aufgabe (10 Punkte, Nutzung mehrdimensionaler Arrays, VL 22)










Zur Veranschaulichung des Ergebnisses ist https://kleuker.iui.hs-osnabrueck.de/querschnittlich/prog1/gameOfLive.html nutzbar. Da das Programm nicht in Java geschrieben ist, sieht das Interaktionsbrett etwas anders aus. Sie können auch ein Feld mit der Wahrscheinlichkeit 0 erstellen, mit der Maus Zellen markieren und sehen was nach einem Schritt passiert.

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 bewohnte 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.

**40. Aufgabe (Exceptions analysieren, VL 23)**

```
public class Analyse {
  public int check(int x)
                throws MeineException {
    EinUndAusgabe io
        = new EinUndAusgabe();
    try {
      io.ausgeben("A");
      if (x < 20) {
        throw new MeineException(x);
      }
      io.ausgeben("B");
      if (x < 62) {
        io.ausgeben("C");
        return x;
      } else {
        io.ausgeben("D");
        throw new MeineException(x-10);
      }
    } catch (MeineException e) {
      try {
        io.ausgeben("E");
        if (e.getStufe() > 72) {
          throw new MeineException(10);
        }
        io.ausgeben("F");
        if (e.getStufe() < 62) {
          io.ausgeben("G");
        } else {
          io.ausgeben("H");
          throw new MeineException(x);
        }
      } finally {
        io.ausgeben("Y");
      }
    } finally {
      io.ausgeben("Z");
    }
    io.ausgeben("I");
    return -42;
  }
}
```

```
public class MeineException extends
Exception {

  private int stufe;

  public MeineException(int wert) {
    super();
    this.stufe = wert;
  }

  public int getStufe() {
    return this.stufe;
  }
}
```

```
public class Main {

  public static void main(
                        String[] args) {
    EinUndAusgabe io
        = new EinUndAusgabe();
    int[] test = {15, 35, 65, 75, 85};
    Analyse a = new Analyse();
    for(int t: test) {
      io.ausgeben(t +": ");
      try {
        io.ausgeben(a.check(t) + "\n");
      } catch (Exception e) {
        io.ausgeben("Schicht\n");
      }
    }
  }
}
```

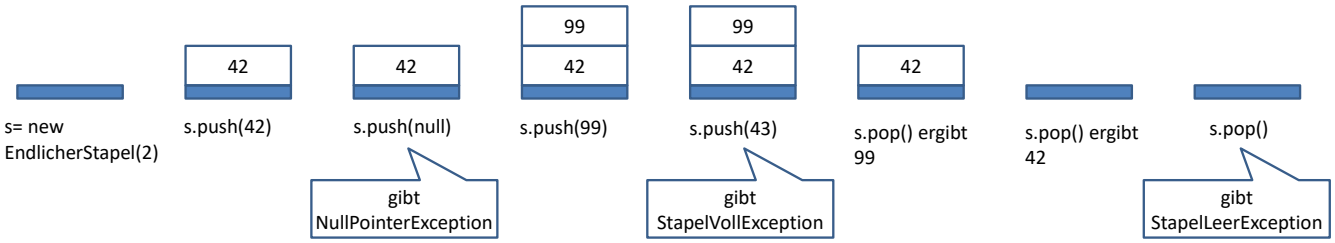
Geben Sie die Ausgaben des obigen Programms an.

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

**Programmierung 1**

Wintersemester 2025/26
**Aufgabenblatt 12**

## 41. Aufgabe (Stapel-Variante, Umgang mit Exceptions, Testen, VL 24)




Ein Stapel (Stack) ist eine Datenstruktur, die Objekte eines bestimmten Typs aufnehmen kann. Dabei werden Objekte mit einer Methode push(.) so auf den Stapel gelegt, dass das neue Element immer oben liegt. Mit der Methode pop(.) wird das oberste Element zurückgegeben und vom Stapel gelöscht. Weiterhin ist es sinnvoll, wenn es Methoden istLeer() bzw. istVoll() gibt, mit denen man prüfen kann, ob ein Stapel voll oder leer ist.

a) Realisieren Sie eine Klasse EndlicherStapel für Integer-Objekte, mit einem Konstruktor, der die mögliche Größe des Stapels als Parameter hat. *Nutzen Sie zur Realisierung des Stapels einen Array*.
b) Realisieren Sie eine Methode push(Integer), mit der ein Integer-Objekt auf den Stapel gelegt wird. Wird versucht, eine null-Referenz auf den Stapel zu legen, soll eine NullPointerException geworfen werden. Wird versucht ein Objekt auf einen vollen Stapel zu legen, soll eine von Ihren neu zu erstellende StapelVollException geworfen werden.
c) Realisieren Sie die Methode pop(), die das zuletzt auf den Stapel gelegte Integer-Objekt zurück gibt und vom Stapel löscht. Wird versucht, ein Objekt von einem leeren Stapel zu nehmen, soll eine von Ihren neu zu erstellende StapelLeerException geworfen werden.
d) Realisieren Sie die Methoden istVoll() und istLeer() zur Überprüfung des Stapelzustands.
e) Schreiben Sie zu allen realisierten Methoden JUnit-Tests, die alle möglichen Ergebnisse testen. Nutzen Sie dazu eine Test Fixture, also mehrere Objekte, mit einem leeren, einem vollen und einem „normal“ gefüllten Stack, auf denen Sie jeweils alle Ihre Methoden ausprobieren.
f) Kopieren Sie Ihr bisheriges Projekt in ein neues Projekt und ändern Sie Ihre Implementierung so ab, dass alle Tests aus e) erfüllt werden, sich aber trotzdem ein Fehler im Programm befindet. Ergänzen Sie dann einen Test, der den eingebauten Fehler findet.

Hinweis: Die Methoden push(.) und pop() sollen die Exceptions nur werfen, nicht selber bearbeiten. In den Tests ist mit try-catch-Blöcken zu prüfen, ob die richtigen Exceptions geworfen werden und die Methoden sonst die korrekten Ergebnisse liefern.