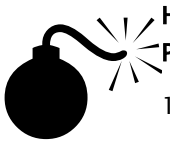


Halten Sie beim Programmieren rigoros ALLE aufgestellten Programmierregeln ein. Ihre Programme werden in Hinsicht auf die Einhaltung dieser Regeln kontrolliert.

1. *Deklarieren* Sie ein Array mit dem Namen `adresse` und *füllen* Sie dieses mit Ihrem *Namen*, der *Straße* und Ihrem *Herkunftsort*:

   `_____ adresse = _____;`

2. Ändern Sie dann die *Straße* auf `Adolf-Munkel-Weg` ab:

   `adresse_____;`

3. Erzeugen Sie ein boolesches Array b, ein **`int`**-Array `i` und ein String-Array `s` mit je *50 Elementen*, *ohne* diesen Elementen einen Wert zuzuweisen:

   ```
   boolean ___ b = _____;
   int ___ i = _____;
   String ___ s = _____;
   ```

4. Angenommen b sei ein instanziiertes Array das viele Elemente enthält. Sie sollen folgende Aufgaben durchführen:

Ausgabe des *zweiten Elementes*	
Ermitteln Sie die *Anzahl der Elemente* im Array	
Ausgabe des *vorletzten Elementes* im Array	
Ausgabe des Elementes, das sich genau in der *Mitte* des Arrays befindet	
Das *erste und letzte Element* im Array soll *vertauscht* werden	

5. Erzeugen Sie das **`int`**-Array d mit *32 Elementen* und füllen Sie dieses durch eine Schleife mit den Werten 1, 3, 7, 15, 31, 63, …

6. Geben Sie obiges Array d mit einer *Schleife* aus.

7. Welchen *Datentyp* haben die Elemente des Array `wasBinIch`? Begründen Sie Ihre Entscheidung. Betrachte dazu das folgende Codefragment:

   ```
   System.out.println("Laenge: " + wasBinIch[i].length());
   ```

8. Erklären Sie ausführlich (Zeile für Zeile) die folgende *Fehlermeldung*:

   ```
   Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 98
           at MyTestProg.test(MyTestProg.java:13)
           at MyTestProg.main(MyTestProg.java:22)
   ```

9. Erzeugen Sie ein *Array* mit dem Namen q, welches folgenden Inhalt hat:

11						
21	22	23	24	25	26	27
7						
41	42	43				

10. Geben Sie dann dieses Feld durch zwei in sich geschachtelte **`for`**-Schleifen aus.

11. Schreiben Sie die Methode `erstelleSchiefesBooleanArray`, der die Anzahl der Zeilen/Spalten des zurückzuliefernden „schiefen“ **`boolean`**-Arrays übergeben wird. Wird beispielsweise 4 übergeben, so soll das zurückzuliefernde Array folgendermaßen aussehen

false			
false	false		
false	false	false	
false	false	false	false

12. Schreiben Sie das Programm `TestArray`, dass ein *20 Elemente* großes **int**-Array mit Zufallszahlen zwischen 1 und 100 erzeugt und anschließend ausgibt. Programmieren Sie für das Ausgeben des Arrays folgende Methode:

```
public static void printIntArray(String msg, int[] a)
```

Das Array soll in Java-Notation ausgegeben werden, und vor dem eigentlichen Array soll der Text des Arguments `msg` stehen. So soll z. B. für `msg = "a = "` der Text `a = {1, 12, 5, … }` ausgegeben werden.

Lagern Sie das Erzeugen des Zufallszahlen-Arrays in die Methode

```
public static int[] randomIntArray(int anzahl, int von, int bis)
```

aus. Das erzeugte Array soll `anzahl` Elemente haben. Die Elemente sollen eine zufällige Größe zwischen `von` und `bis` besitzen – `von` und `bis` jeweils inklusive.

Testen Sie die Methode `randomIntArray` mit:

```
int[] a = randomIntArray(5, -5, 20);
printIntArray("a = ", a);
printIntArray("", randomIntArray(50, -1, 1));
```

Schreiben Sie die drei Methoden

```
public static int getMinimum(int[] a)
public static int getMaximum(int[] a)
public static double getMittelwert(int[] a)
```

Testen Sie mit dem Array `randomIntArray(5, -5, 20)`.

Schreiben Sie die Methode

```
public static int indexOf(int[] a, int z)
```

welche die Zahl `z` im Array `a` sucht und die *Position* zurückgibt an welcher die Zahl zum ersten Mal vorkommt. Falls `z` nicht gefunden werden kann, soll `-1` zurückgegeben werden. Testen Sie die Methode…

Schreiben Sie die Methode:

```
public static int indexOf(int[] a, int z, int pos)
```

welche die Zahl `z` im Array `a` ab der Position `pos` sucht und die Position zurückgibt. Falls `z` nicht gefunden werden kann, soll `-1` zurückgegeben werden.

Schreiben Sie die Methode

```
public static int getMinPos(int[] a, int pos)
```

welche im Array `a` ab der Position `pos` inklusive die *Position* des Minimums zurückgibt. Testen Sie mit `a = randomIntArray(5, 1, 10)` und `getMinPos(a, 2)`.

Schreiben Sie die Methode:

```
public static void addZahl(int[] a, int z)
```

die jedes Element von `a` um die Zahl `z` erhöht. Testen Sie mit

```
a = randomIntArray(10, 0, 9);
printIntArray("a = ", a);
addZahl(a, 100);
printIntArray("a + 100 = ", a);
```

Schreiben Sie die Methode

```
public static void swap(int[] a, int i, int j)
```

die das Element `a[i]` mit dem Element `a[j]` vertauscht. Testen Sie mit `a = randomIntArray(3, 1, 10)` und `swap(a, 0, 2)`.

Schreiben Sie die Methode

```
public static void sortMinArray(int[] a)
```

Verwenden Sie hierzu folgenden Sortieralgorithmus:

Suchen Sie mit der Methode `getMinPos` die Position des kleinsten Elements im gesamten Array und vertauschen Sie es mit dem ersten Element. Verwenden Sie beim Vertauschen die Methode swap. Danach suchen Sie ab dem 2. Element das Kleinste und vertauschen es mit dem 2. Element. Hierzu setzen Sie wieder die Methoden `getMinPos` und swap ein. Dies wiederholen Sie bis zur vorletzten Position im Feld (**BEMERKUNG:** Diese Art der Sortierung wird als *„Sortieren durch Minimumsuche"* bezeichnet).

Testen Sie die Methode indem Sie das Array `randomIntArray(50, 1, 100000)` sortieren.

Schreiben Sie die Methode

```
public static int[] delDoppelte(int[] a)
```

die aus dem Array a alle doppelten Zahlen löscht und ein neues, von doppelt vorkommenden Zahlen gesäubertes Array zurück gibt. Wird beispielsweise die Methode auf das Array 1, 3, 3, 1, 2, 1, 5 angewendet, so liefert sie das Array 1, 3, 2, 5 mit der Länge `length = 4` zurück. Das ursprüngliche Array a darf durch die Methode nicht verändert werden.

Testen Sie mit

```
a = randomIntArray(50, 0, 10);
printIntArray("a = ", a);
a = delDoppelte(a);
printIntArray("adeldoppelte = ", a);
```

13. Ein bekannter *Kartentrick* funktioniert folgendermaßen:

    a. Vom Spieler A werden 21 Spielkarten in 3 Spalten zu je 7 Karten aufgelegt.

    b. Spieler B wählt eine Karte und teilt A die Spalte mit, in der die ausgewählte Karte liegt.

    c. Spieler A nimmt die Karten jeweils Spalte nach Spalte auf, die von B bezeichnete Spalte als zweite und legt sie reihenweise hin. B teilt wieder die Spalte mit.

    d. Wie c.

    e. A legt die Karten nochmals wie in c. auf – die von B ausgesuchte Karte ist in der zweiten Spalte die vierte.

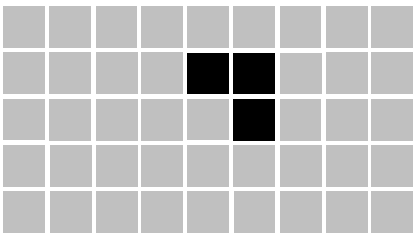
    Schreiben Sie das Programm mit dem Namen `Kartentrick`, das die Rolle des Spielers A übernimmt und den Kartentrick beliebig oft ausführt. Die `main`-Methode muss dabei folgende Struktur haben:

```
do {
   int[][]karten = null;
   karten = fuellen(karten, 0);
   ausgeben(karten);
   int spalte = readInt("Spalte der Karte: ");

   karten = fuellen(karten, spalte);
   ausgeben(karten);
   spalte = readInt("Spalte der Karte: ");

   karten = fuellen(karten, spalte);
   ausgeben(karten);
   spalte = readInt("Spalte der Karte: ");

   karten = fuellen(karten, spalte);
   ausgeben(karten);
   System.out.println("Karte " + karten[3][1] +
      " wurde gewählt");

} while (Character.toLowerCase(
   readChar("Nochmals (j/n)? ")) == 'j');
```

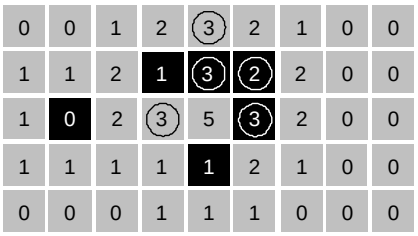
    Programmieren Sie die Methoden `fuellen` und `ausgeben`. Überlegen Sie sich vorher die geeigneten *Parameterlisten* und *Rückgabewerte*, beschreiben Sie in den *Kommentaren*, wie die Methoden funktionieren sollen.

```
Kartentrick
===========
  1  8 15
  2  9 16
  3 10 17
  4 11 18
  5 12 19
  6 13 20
  7 14 21
Spalte der Karte: 2
  1  2  3
  4  5  6
  7  8  9
 10 11 12
 13 14 15
 16 17 18
 19 20 21
Spalte der Karte: 1
  3  6  9
 12 15 18
 21  1  4
  7 10 13
 16 19  2
  5  8 11
 14 17 20
Spalte der Karte: 3
  6 15  1
 10 19  8
 17  9 18
  4 13  2
 11 20  3
 12 21  7
 16  5 14
Karte 13 wurde gewählt
Nochmals (j/n)? n
```

14. Das **Spiel des Lebens** (*engl. Conway's Game of Life*) ist ein vom Mathematiker John Horton Conway 1970 entworfenes System zweidimensional angeordneter zellulärer Automaten[1]. In jedem Automaten bzw. Feld kann Leben sein oder nicht.



Schwarz: Lebendes Feld
Grau: Totes Feld



Nach folgenden Regeln entsteht oder vergeht Leben in einem Feld:

- Ein lebendes Feld bleibt lebend, wenn es genau 2 oder 3 lebende Nachbarfelder hat, ansonsten stirbt es.
- Neues Leben entsteht, wenn ein totes Feld genau 3 lebende Nachbarfelder hat.
- Hat ein lebendes Feld weniger als 2 lebende Nachbarfelder, dann geht es aus Einsamkeit zugrunde.
- Hat ein lebendes Feld mehr als 3 lebende Nachbarfelder, dann stirbt es wegen Überbevölkerung.

Die Zahlen im nebenstehenden Bild geben die Anzahl der lebenden Nachbarfelder an. Die Kreise geben an, ob das Feld im nächsten Lebenszyklus lebt.

Schreiben Sie ein Programm mit dem Namen GameOfLife, welches die Lebenszyklen simuliert und am Bildschirm ausgibt. Das Programm soll so programmiert werden, dass die Größe der Matrix über Konstanten festgelegt werden kann. Die Konstanten selbst können bei Bedarf geändert werden, so dass das Programm einmal mit wenigen Feldern und dann wieder mit vielen Feldern arbeiten kann:

```
// Fixe Anzahl von Zeilen und Spalten
final int ANZAHL_ZEILEN = 10;
final int ANZAHL_SPALTEN = 10;
// Maximale Anzahl von Iterationsschritten
final int MAX_SCHRITTE = 150;

boolean[][] matrix1 = new boolean[ANZAHL_ZEILEN][ANZAHL_SPALTEN];
```

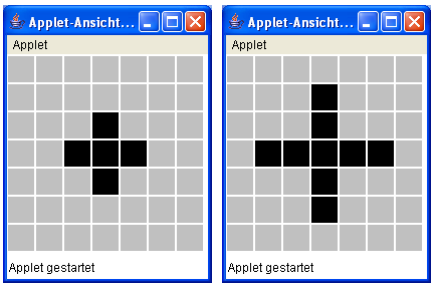
Das Hauptprogramm sollte die nebenstehende Struktur haben. Um diese zu realisieren, benötigen Sie folgende Methoden, die Sie zuerst in ihren Parameterköpfen und Rückgabewerten exakt definieren, dann ausprogrammieren und testen sollten

Fülle Leben in die Matrix1
Initialisiere Schrittzähler
Gib Matrix1 am Bildschirm aus
Matrix2 soll sein Matrix1
Berechne das Leben in Matrix1 neu
Erhöhe Schrittzähler
Wiederhole solange Schrittzähler kleiner MAX_SCHRITTE und Unterschiede zwischen Matrix1 und Matrix2

ACHTUNG: Sie dürfen nur jene Parameter den nachfolgenden Methoden übergeben, welche in den Beschreibungen angeführt werden.

Methode `fuellenMatrixSternMitte`

Diese Methode füllt in die übergebene Matrix einen Stern, der eine bestimmte Größe hat. Die Größe des einzufügenden Sterns wird der Methode übergeben. Der Stern wird genau in die Mitte der Matrix eingefügt. Wird für die Sterngröße 0 übergeben, so wird ein einziges Feld in der Mitte der Matrix gefüllt. Nebenstehend sehen Sie Sterne der Größe 1 und 2, welche in die Matrix gefüllt wurden.



Um diese Methode effizient testen zu können, sollten Sie zuerst die nächste Methode ausprogrammieren.

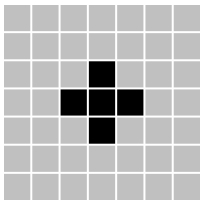
[1] Auszug aus http://www.wikipedia.de

Methode ausgebenMatrix

Durch diese Methode wird die ihr übergebene Matrix am Bildschirm ausgegeben. Zur Bildschirmausgabe benötigen Sie die Möglichkeit auf einer Zeichenfläche (engl. Graphiccontext) Rechtecke in verschiedenen Farben zu positionieren. Das bereitgestellte Java-Applet mit dem Namen TestGrafikAusgabe zeigt Ihnen, wie Sie über ein Applet eine Zeichenfläche erstellen, ansprechen und dort Linien und Rechtecke in verschiedenen Farben ausgeben können. Verwenden Sie dieses Programm als Grundlage für Ihr Programm, und schreiben Sie dieses um.

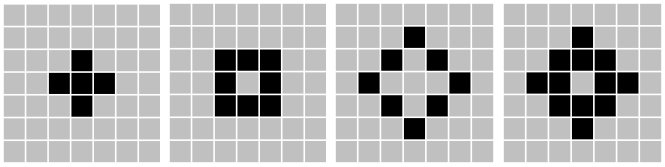
Damit Ihre Methode auf der Zeichenfläche zeichnen kann, müssen Sie ihr diese Zeichenfläche in einem Parameter übergeben (`Graphics g`). Programmieren Sie die Methode so, dass sich die Größe der ausgegebenen Rechtecke auf die Größe der Zeichenfläche anpassen.

Methode anzahlLebendeNachbaren



Diese Methode ermittelt in der übergebenen Matrix, wie viele lebende Nachbaren ein Feld hat. Das zu analysierende Feld wird durch seine Zeilen- und Spaltenposition übergeben. So liefert diese Methode für die nebenstehende Matrix für die Position 3, 3 den Wert 4 zurück, für die Position 2, 2 wird 3 zurück geliefert.

Methode berechneMatrix



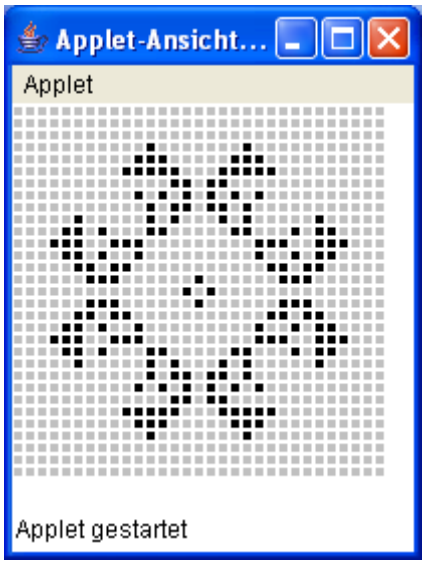
Diese Methode verwendet die vorige Methode, um einen neuen Lebenszyklus für die ihr übergebene Matrix zu berechnen. Das neu berechnete Leben wird in einer neuen Matrix zurück geliefert. Dabei wird die alte Matrix nicht geändert. Nebenstehend sehen Sie vier Lebenszyklen die aufeinanderfolgend entstehen.

Methode existierenUnterschiede

Diese Methode kontrolliert ob die beiden ihr übergebenen Matrizen unterschiedlich sind oder nicht. Sie liefert **true** zurück, falls die Matrizen an unterschiedlichen Stellen gefüllt sind.

Damit Sie die Geschwindigkeit in der die Simulation abläuft einstellen können, sollen Sie in Ihr Programm folgende Methode einfügen und in der Schleife aufrufen:

```
/**
 * Veranlasst dass das Programm millis Millisekunden
pausiert
 * @param millis Anzahl der Millisekunden die
gewartet werden
 */
public void bremse(int millis) {
   try {
      Thread.sleep(millis);
   } catch (InterruptedException e) {
   }
}
```



Testen Sie dann Ihr Programm mit größeren Matrizen und größeren Sternen aus, beobachten Sie, wie sich Leben in den Matrizen entwickelt…

15. Schreiben Sie noch die Methode `fuellenMatrixZufaellig`, welche in die ihr übergebene Matrix an zufälligen Positionen Leben einfügt. Der Methode wird auch ein Verhältnis übergeben, das ihr sagt wie das Verhältnis zwischen gefüllten und nicht gefüllten Positionen sein soll. Das Verhältnis 0.1 bedeutet, dass 10% der Positionen mit Leben gefüllt werden sollen.

    Probieren Sie dann Ihr Programm mit mehreren zufällig gefüllten Matrizen aus und versuchen Sie zu verstehen wie das Verhältnissen das Leben in der Matrix beeinflusst…