Java02 − „Feld“ bzw. „array“: Zufallswerte und Suche nach Minimum

Schritt 1 − BlueJ-Projekt öffnen

Öffne Dein BlueJ-Projekt „**Java_02_Feld**“ der letzten Stunde.

Schritt 2 − Methode „`feldAusgeben()`“ zur Ausgabe der Inhalte aller Feld-Elemente

Wir wollen eine Methode erstellen, bei deren Aufruf der gesamte Feld-Inhalt ausgegeben wird, d. h. die Inhalte aller Feld-Elemente. Das gelingt mit einer Wiederholung mit fester Anzahl („for-Schleife“) unter Verwendung der bereits vorhandenen Methode „`feldElementIAusgeben(int i)`“ (letzte Stunde):

```
public void feldAusgeben() {
    for(int i = 0; i < feld.length; i++) {
        feldElementIAusgeben(i);
    }
}
```

Der Index `i` der for-Schleife beginnt mit 0 (dem Index des ersten Feld-Elements). Die for-Schleife läuft solange, wie `i` kleiner ist als die Länge des Feldes „feld“, also `i < feld.length` .
Man hätte für unser Beispiel-Feld der Länge 5 auch `i < 5` schreiben können. Aber so ist die Methode geeignet für die Ausgabe der Inhalte aller Feld-Elemente von Feldern beliebiger Länge.

Teste die Methode „`feldAusgeben()`“ durch Aufruf auf einem Beispiel-Feld!

Schritt 3 − Konstruktor zur Erzeugung eines Feldes beliebiger Länge mit Zufallszahlen

Da wir die Feld-Elemente mit Zufallszahlen füllen möchten, müssen wir − noch vor der Definition der Klasse − ein „Hilfs-Paket“ importieren. Die entsprechende Angabe steht in Zeile 1 des Quellcodes:

```
import java.util.Random;        // Import des Hilfs-Pakets für Zufallszahlen
public class Feld { …      // schon vorhanden von letzter Stunde
```

Direkt unterhalb des bereits vorhanden Konstruktors „`public Feld(int a1, int a2, …)`“ (letzte Std.) fügen wir einen weiteren Konstruktor „`public Feld(int feldlaenge, int grenze)`“ ein.
Der Name „Feld“ dieses Konstruktors entspricht − wie immer − dem Namen „Feld“ der Klasse.
Er unterscheidet sich vom Konstruktor „`public Feld(int a1, int a2, …)`“ (letzte Std.) durch andere Parameter, die beim Aufruf abgefragt werden und dann bei der Objekt-Erzeugung „verarbeitet“ werden.
Man sagt: Der zweite Konstruktor hat eine andere Signatur als der erste Konstruktor.

Durch den Parameter „`feldlaenge`“ legen wir die Länge des gewünschten Feldes fest. Durch den Parameter „`grenze`“ wird festgelegt, bis zu welcher größten Zahl die Zufallszahlen erzeugt werden.
Bsp: Der Wert „6“ für „`grenze`“ bewirkt, dass Zufallszahlen von 1 bis 6 erzeugt werden (z. B. für „Würfeln“).

Hier der vollständige zweite Konstruktor „`public Feld(int feldlaenge, int grenze)`“:

```
public Feld(int feldlaenge, int grenze) {      // Zweiter Konstruktor
    feld = new int[feldlaenge];      // Festlegung der Feld-Länge
    Random zufallszahl = new Random();      // Zufallszahl-Erzeugung
    for(int i = 0; i < feldlaenge; i++) {      // i = 0 bis feldlaenge-1
        feld[i] = zufallszahl.nextInt(grenze)+1;      // „Befüllung“
    }
}
```

Erzeuge ein Feld der Länge 20 mit Zufallszahlen von 1 bis 6 und rufe darauf „`feldAusgeben()`“ auf!

FORTSETZUNG AUF SEITE 2

Schritt 4 – Methode „`findeMinimum()`“ zum Finden des Minimums und seiner Position

Kopiere die Methode „`findeMinimum()`“ ganz unten in Deinen Quellcode (aber noch oberhalb der schließenden „Klassen-Klammer“).

Die hier erscheinenden Zeilenumbrüche bei ** musst Du löschen!

```
    public void findeMinimum() {
        int min = feld[0];      // Annahme: Minimal-Wert ist feld[0]
        int minPos = 0;         // Annahme: Minimum-Position ist 0
        for (int i = 1; i < feld.length; i++) {     // Feld durchlaufen
            if (feld[i] < min) {      // wenn feld[i] „noch kleiner“
                min = feld[i];        // dann ist feld[i] das neue Minimum
                minPos = i;           // und i die neue Minimum-Position
            }
        }
        System.out.println("Das Element mit dem Index " + minPos + " hat den
** kleinsten Wert: " + min + ".");
    }
```

„Übersetze“ den Quellcode.
Erzeuge mit dem zweiten Konstruktor „`public Feld(int feldlaenge, int grenze)`“ ein Feld der Länge 10 mit Zufallszahlen von 1 bis 1000.
Lasse Dir mit „`feldAusgeben()`“ zunächst das gesamte Feld anzeigen und suche das Minimum.
Rufe nun auf diesem Feld die Methode „`findeMinimum()`“ auf. Tut die Methode, was sie soll?

ZUSATZ-AUFGABEN

a) **Methode „`durchschnitt`“ zur Ermittlung des Durschnitts der Feld-Element-Werte**

Tipp: Beginne so:

```
    public void durchschnitt() {
        double summe = 0;      // „double“, damit „Kommazahlen“ „rauskommen“
        double durchschnitt = 0;
```

Teste die Methode „`durchschnitt`“ durch Aufruf auf einem zuvor erzeugten Feld großer Länge (z. B. 6000 oder 60 000 oder 600 000 oder 6 000 000 oder 60 000 000) mit Zufallszahlen von 1 bis 6. Das entspricht einem 6000-maligen oder … Würfeln.
Welchen Durschnitt wird man erwarten?

b) **Methode „`werteZaehlen`“ zur Ermittlung der Anzahl der Feld-Elemente mit einem best. Wert**

Wir bleiben beim Würfeln, d. h. die Methode bezieht sich explizit nur auf Felder mit ganzzahligen Feld-Element-Werten von 1 bis 6.
Es soll gezählt und am Ende ausgegeben werden, wie oft die Zahlen 1 bis 6 jeweils „gewürfelt“ wurden.
Tipp: Beginne so:

```
    public void werteZaehlen() {
        int anzahl1 = 0;      // lokale Variable zum Zählen der „Einser“
        int anzahl2 = 0;
        …
        for (int i = 0; i < feld.length; i++) {
            if(feld[i] == 1) {
                anzahl1 = anzahl1 + 1;
            } else if(feld[i] == 2) {
                anzahl2 = anzahl2 + 1;
            } …
        }
        System.out.println("Einser: " + anzahl1);
        …
    }
```

Teste die Methode „`werteZaehlen`“ durch Aufruf auf einem zuvor erzeugten Feld großer Länge (z. B. 6000 oder 60 000 oder 600 000 oder 6 000 000 oder 60 000 000) mit Zufallszahlen von 1 bis 6.