technische universität
dortmund

Fakultät für Informatik
Lehrstuhl 14
Lars Hildebrand

Praktikum zu

Einführung in die Informatik für LogWilngs und WiMas

Wintersemester 2017/18

Übungsblatt 8

Besprechung:
11.–15.12.2017
(KW 50)

Vorbereitende Aufgaben

Dieses Übungsblatt beschäftigt sich mit **Arrays** und Sortieralgorithmen.

Aufgabe 8.1: Manuelle Sortierung

Gegeben ist folgendes Array:

7	12	3	2	21	9

In der Vorlesung (Kapitel 5.1) haben Sie einen Algorithmus zum Sortieren von Arrays kennengelernt. Dieser heißt **Selectionsort**. Sortieren Sie das Array nun aufsteigend mit dem Selectionsort-Verfahren. Notieren Sie dabei jede **Tauschoperation**:

Aufgabe 8.2: Debugging

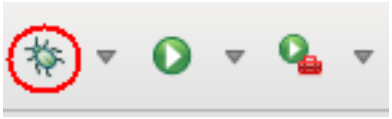
Lesen und verstehen Sie den Text über Debugging auf der nächsten Seite.

Debugging

Mit Hilfe eines sog. Debuggers können Sie ein Programm zur Laufzeit analysieren. Eclipse bietet eine sehr einfache Möglichkeit, dies umzusetzen.

Sie können mit Hilfe von sog. **Breakpoints** Zeilen im Code angeben, an denen das Programm **vor** seiner Ausführung anhalten soll. Dazu müssen Sie links auf den Zeilenrand Ihres Quellcodes doppelklicken, so dass ein kleiner, blauer Punkt erscheint. Sie können den Breakpoint auf die selbe Art und Weise wieder entfernen.

Der Eclipse-Debugger kann dann durch das Klicken auf den kleinen, grünen Käfer neben dem Run-Button gestartet werden.



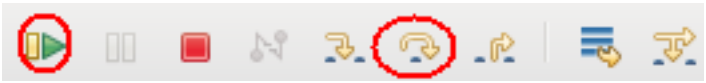
Debug-Button

Wenn Sie das Programm über den Debugger starten, wird Eclipse seine Ansicht wechseln wollen, sobald ein Breakpoint erreicht wird. Akzeptieren Sie diesen Vorschlag.

Ihr Quellcode wird in den unteren Teil des Bildschirms verschoben und in der oberen Hälfte des Bildschirms erscheinen Werkzeuge zum Analysieren Ihres Programmes. Zudem pausiert die Ausführung des Programmes am erreichten Breakpoint. Sie können sich in der oberen, rechten Tabelle alle lokalen Variablen mit ihren Werten ansehen.

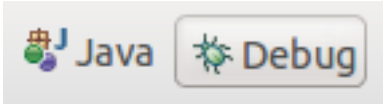
Im oberen Bereich ihrer Arbeitsumgebung können Sie durch das Klicken des grünen Pfeiles in Form eines „Abspielen“-Symboles den Programmfluss bis zum Erreichen des nächsten Breakpoints fortsetzen.

Durch Klicken des sich rechts daneben befindenden Buttons mit dem Tooltip `Step Over(F6)` können Sie das Programm genau eine Anweisung durchführen lassen, um so schrittweise den Ablauf des Programmes zu beobachten.



zum nächsten Breakpoint (links), schrittweise Ausführung (rechts)

Wenn Sie zurück zur normalen Editor-Ansicht wechseln wollen, gibt es in der oberen, rechten Ecke des Bildschirms einen dafür vorgesehenen Reiter.



Umschalten der Editor-Ansicht

Präsenzaufgaben

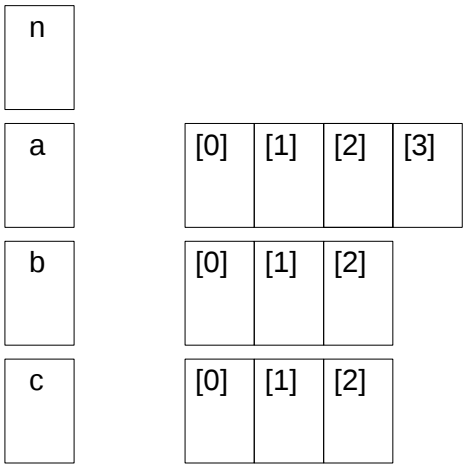
Aufgabe 8.3: Speicherverwaltung für Felder

In dieser Aufgabe sollen Sie sich mit der Speicherverwaltung im Zusammenhang mit Arrays beschäftigen.

Betrachten Sie das untenstehende Programmfragment. Es deklariert und initialisiert eine ganzzahlige Variable **n** und drei Felder von ganzen Zahlen **a**, **b** und **c**.

Gehen Sie den Programmtext nun Zeile für Zeile durch und tragen Sie in das nebenstehende Diagramm ein, wie die einzelnen Speicherzellen im Laufe des Programms belegt und geändert werden.

```
1  int[] a;
2  a = new int[4];
3  for (int i = 0; i < a.length; i++) {
4      a[i] = i;
5  }
6  int n = a.length - 1;
7  int[] b;
8  b = new int[n];
9  for (int i = 0; i < b.length; i++) {
10     b[i] = a[i] + 10;
11 }
12 int[] c = new int[b.length];
13 c = b;
```



Aufgabe 8.4: Arrays als Parameter

In dieser Aufgabe sollen Sie lernen, Arrays als Parameter zu nutzen. Legen Sie dazu eine Klasse **UseArrays** an.

- Schreiben Sie eine Funktion **printArray**, die ein übergebenes **int**-Array ausgibt.
- Implementieren Sie eine Funktion mit dem Namen **swap**, die keinen Wert zurückgibt und ein **int**-Array, sowie zwei Indizes entgegen nimmt und die Elemente im Array an den entsprechenden Indizes miteinander tauscht.
- Verwenden Sie anschließend die Funktion **swap** um das erste mit dem letzten Element zu tauschen. Geben Sie das Array vor und nach dem Aufruf von **swap** mit der Funktion **printArray** aus.
- Was fällt Ihnen auf, wenn Sie das Ergebnis mit den Erkenntnissen aus Aufgabe 6.4 vergleichen?

Aufgabe 8.5: Bibliothek erstellen

In dieser Aufgabe sollen Sie eine Klasse programmieren, mit der Sie Arrays mit verschiedenen Eigenschaften erzeugen können.

- Erstellen Sie eine neue Klasse namens **ArrayGenerator** im Paket **blatt08**. Diese Klasse soll **keine main**-Methode erhalten!
- Implementieren Sie die folgenden beiden Funktionen in dieser Klasse:

```
public static int[] generateAscendingArray(int length) {...}
public static int[] generateRandomArray(int length) {...}
```

- Die Funktionen **generateAscendingArray** soll ein **int**-Array erzeugen und es aufsteigend mit den Werten von 1 bis **length** befüllen.
- Die Funktion **generateRandomArray** soll ein zufällig gefülltes Array mit der Länge **length** erzeugen. Dafür müssen Sie die Klasse **Random** aus dem Paket **java.util** importieren, so wie Sie es bereits mit dem **Scanner** getan haben:
  - Mit `Random randomNumbers = new Random();` können Sie einen neuen Zufallsgenerator instanziieren.
  - Mit `randomNumbers.nextInt(bound)` können Sie danach eine zufällige Zahl aus dem halboffenen Intervall $[0, \textbf{bound})$ erhalten.
  - Verwenden Sie als **bound** die Länge des erzeugten Arrays.

Aufgabe 8.6: Bibliothek verwenden

In dieser Aufgabe sollen Sie die Funktionen der soeben definierten Klasse verwenden.

Eine Klasse ohne **main**-Methode, wie Ihre **ArrayGenerator**-Klasse, nennt man auch **Bibliothek**. Sie ist ein Ort, an dem man hilfreiche Funktionen finden kann.

- Verwenden Sie in der **main**-Methode der Klasse **UseArrays** die soeben von Ihnen definierten Funktionen, um ein aufsteigendes und ein zufälliges Array der Länge 32 auszugeben.
- Schreiben Sie die Funktion **arrayMin**, die das Minimum eines übergebenen **int**-Arrays zurückgibt. Rufen Sie diese Funktion für die generierten Arrays auf.
- Schreiben Sie die Funktion **average**, die den Mittelwert eines übergebenen **int**-Arrays berechnet und als **double** zurückgibt. Rufen Sie auch diese Funktion für die generierten Arrays auf.

Ergänzende Aufgaben

Aufgabe 8.7: Zeitmessung

In den vergangenen Aufgaben haben Sie die Fibonacci-Folge bereits kennengelernt. Hier wollen wir verschiedene Implementierungen vergleichen.

a) Überlegen Sie sich eine rekursive Definition der Fibonacci-Folge.

b) Implementieren Sie in der Klasse **FibonacciComparison** eine Funktion **recFib**, die einen **int**-Wert **n** entgegen nimmt, die n-te Fibonacci-Zahl rekursiv berechnet und diese als **long** zurückgibt.

c) Machen Sie aus Ihrer iterativen Implementierung der Fibonacci-Folge aus Blatt 5 eine Funktion **itFib** mit den selben Eigenschaften.

d) Diese Implementierungen vergleichen wir nun miteinander:

- Die Funktion **System.nanoTime()** gibt einen Zeitstempel als **long**-Wert zurück. Die Differenz zweier solcher Zeitstempel ist die zwischen den Aufrufen vergangene Zeit in Nanosekunden. Diese können wir verwenden, um Laufzeiten von Programmabschnitten zu messen.
- Rufen Sie in der **main**-Funktion die rekursive und die iterative Berechnungsfunktion mit dem selben Wert auf.
- Messen Sie mit jeweils zwei Zeitstempeln die Laufzeit der jeweiligen Implementierung.
- Was fällt Ihnen auf? Was könnten Vor- und Nachteile der jeweiligen Implementierungen sein?