Programmieren in Java:
Visualisierung von Sortieralgorithmen

von
Kai Richard König (12.03)

07 2011

FOS/BOS Technik München

Zusammenfassung

Dieses Dokument gibt Aufschluss über die Überlegungen hinter der Entwicklung sowie der Programmierung einer Java-Swing Desktop-Application zur Visualisierung von Sortieralgorithmen.

Inhaltsverzeichnis

Kapitel 1
Grobkonzept

1.1 Was macht das Programm ?

Das Java-Programm soll die Funktionsweise von fünf unterschiedllichen Sortieralgorithmen[1] mit Hilfe von Diagrammen anschaulich darstellen. Zur Verdeutlichung der Vorgänge wird jedem Algorithmus ein Balken-Diagramme zugewiesen, in dem jeder Zahlenwert einem Balken entspricht. Vergleicht der Algorithmus nun zwei Zahlenwerte miteinander, so werden die korrespondierenden Balken farblich hervorgehoben. Entscheidet der Algorithmus das einer der Balken vorschoben werden muss um alle Zahlenwerte in eine korrekte, aufsteigende Reihenfolge zu bringen, so wird auch diese Operation im Balken-Diagramm durch farbliche Hervorhebung visualisiert.

Um die fünf dargestellten Algorithmen besser vergleichen zu können, laufen alle Operationen synchronisiert und parallel ab. Zudem kann jeder Algorithmus pro Zeiteinheit nur eine Änderung an den Zahlenwerten durchführen. Diese Restriktion stellt sicher, dass die Ergebnisse reproduzierbar bleiben. Zusätzlich hat der Benutzer die Möglichkeit spezielle Zahlenwert/Balken-Kombinationen auszuwählen um die Schwächen und Stärken einzelner Algorithmen deutlich sichtbarer zu gestalten. Möglich Kombinationen sind zum Beispiel:

- Zahlenwerte, die in mathematisch umgekehrter Reihenfolge vorliegen
- Zahlenreihen, in denen es nur eine geringe Anzahl an Zahlenwerten an der „falschen“ Stelle gibt
- Zahlenreihen, bei denen es mehrere gleiche Zahlenwerte gibt

Wichtig ist vorallem, dass dem Anwender bei der Betrachtung klar wird welcher Algorithmus sich für welche Problemstellung am besten eignet. Erreicht wird dies durch ein Liniendiagramm welches am Ende der Laufzeit eines jedes Algorithmus über das zugewiesene Balken-Diagramm gelegt wird. Durch den Vergleich der Linien ist ersichtlich, wie sich der Grad der SSortiertheit”bei den verschiedenen Algorithmen über die Zeit entwickelt. Die Anzahl der durchgeführten Operationen ist unterhalb der Liniendiagramme angegeben.

[1] Bubble sort(Type: Exchange sort) — Quicksort(Type: Exchange sort) — Heapsort(Type: Selection sort) — Insertion sort(Type: Insertion sort) — Merge sort(Type: Merge sort) —

Kapitel 2
Fachkonzept

2.1 Begriffs- und Notationsklärung

2.1.1 Big-O-Notation

$$O(n)$$

Diese Notation dient zur Angabe der Effizienz von Algorithmen. Sie gibt die maximale Anzahl der Operationsschritte im schlechtesten, durschnittlichen oder besten Fall an (Worst/Average/Best-Case-Scenario).
Beispielsweise bedeutet $O(n^2)$, dass der Algorithmus bei einem Array von 4 Integern maximal 16 Operationschritte benötigt bis das Array sortiert ist - oder weniger da $4^2 = 16$. Von dieser Notation wird im Folgenenden mehrere Male gebrauch gemacht um die Algorithmen zu beschreiben. Mathematisch korrekter wäre jedoch $O(f(n^2))$, da der Variable n noch ein konstanter Faktor für den Algorithmus vorsteht.

2.1.2 Die Applikation oder App

Mit der „Applikation“ oder „App“ wird im Folgenden die von mir entwickelte Java-Desktop-Application zur Visualisierung von Sortieralgorithmen bezeichnet. Die Applikation liegt in kompilierter und unkompilierter Form vor und ist auf der beigelegt CD-ROM zufinden.

2.1.3 Algorithmus

> *“ Ein Algorithmus ist eine aus endlich vielen Schritten bestehende eindeutige Handlungsvorschrift zur Lösung eines Problems oder einer Klasse von Problemen. ”(Wikipedia,* Algorithmus*)*

2.2 Visualisierungen von Algorithmen

Informationen können von unserem Gehirn grundsätlich sehr viel schneller in bildlicher Form aufgenommen und ausgewertet werden, als in Textform. Darum macht es Sinn die Abläufe eines Sortieralgorithmus zu visualisieren anstatt den den Vorgang in schriftlicher Form oder gar als Programmcode zu beschreiben. Zudem sind Sortieralgorithmen ein wichtger Bestandteil in Informatik-Studiengängen. Dort könnte eine Möglichkeit zum besseren Versändnis von Sortieralgorithmen eine zusätzliche optische Bertachtung darstellen. In den folgenden Abschnitten werden nun die fünf in der Applikation verwendeten Sortieralgorithmen vorgestellt und miteinander verglichen.

2.3 Verwendete Algorithmen

2.3.1 Bubblesort

Der Bubblesort-Algorithmus ist der einfachste unter den Sortieralgorithmen. Er ist dementsprechend langsam, da für den Algorithmus ein wenig weitsichtiger Ansatz gewählt wurde. Jedes Element eines Arrays wird mit dem direkt nachfolgenden Element verglichen. Ist das erste Element größer, so tauschen beide Element ihren Platz. Dieser Vorgang wird solange wiederholt bis keines der Element mehr den Platz wechselt. Diese Herrangehensweise impliziert, dass die Prüfung ob ein Array bereits sortiert ist, schon eingebaut ist. Ebenso ergibt diese Handlungsvorschrift eine 'avarage-case performance' von $O(n^2)$. Würde Bubblesort nun versuchen ein schon sortiertes Array zu sortieren, so bräuchte der Algorithmus genau die Anzahl Schritte, die das Array groß wäre. Daraus würde sich dann eine 'best-case performance' von $O(n)$ ergeben (vgl. H. W. Lang, *Sortierverfahren Bubblesort*).

2.3.2 Quicksort

Quicksort ist ein sehr schneller Sortieralgorithmus mit einer average-case performance von $O(n \cdot log(n))$. Funktionsweise und Aufbau sind relativ simpel gehalten. Die gute Performance wird durch eine Verfahren erreicht das sich „Teile und herrsche“ (lat. Divide et impera!, engl. Divide and conquer)Wikipedia, *Quicksort* nennt. Dabei wird das Problem in kleinere Problemeinheiten unterteilt, diese werden dann gelöst und am Ende wieder zusammengeführt. Der Sortieralgorithmus wählt zunächst ein Orientierungselement aus, in den meisten fällen bietet sich dazu das Element in der Mitte an. Nun werden alle Element die kleiner sind als das Orientierungselement nach links und jene die größer sind nach rechts verschoben. Anschließend wird der Vorgang mit den Unterprobleme solange wiederholt - der Algorithmus wird rekursiv aufgerufen

- bis die Länge der auftretenden Unterprobleme den Wert zwei erreicht hat. In der Praxis erweist sich ein gut implentierter Quicksort-Algorithmus als das schnelltste Sortierverfahren (vgl. H. W. Lang, *Sortierverfahren Quicksort*).

2.3.3 Heapsort

Heapsort trägt seinen Namen, da er ein Problem zuerst in einen Heap überführt und anschließend sortieren. Heap bedeutet im Englischen „der Haufen“ oder auch „die Menge“, eigenlich ist damit jedoch ein Binärbaum (engl. Binary Tree, Binary Heap) gemeint. Ein Binärbaum ist ein Datenstruktur bei der es sogenannte Knoten (engl. leafs, nodes) gibt, die maximal zwei Unterknoten haben können. Diese Struktur enthält Eigenschaften, die der Sortieralgorithmus für sich nutzt. Der Binary Heap wird von links nach rechts und von oben nach unten gefüllt. Zunächst wird eine sogenannte Wurzel (engl. root node) angelegt, diese Wurzel bekommt dann zwei Knoten und so weiter bis das Problem komplett in dem Binärbaum präsent ist. Zugleich wird ein Regelwerk umgesetzt das besagt, dass ein Knoten keinen Unterknoten besitzen darf, dessen Wert größer ist als der eigene. Folglich steht an der Wurzel des Binärbaumes immer der größte Wert. Wird dieser Wert nun entnommen und an das Ende einer neuen Datenstruktur oder eines Arrays geschrieben, so wird der BinärBaum nach dem beschrieben Regelwerk neugeordnet. Jetzt steht wieder das größte Element an der Wurzel, so dass es erneut entnommen werden kann. Diesen Vorgang wiederholt Heapsort bis der Binary Heap komplett abgebaut ist. Im Ganzen ergibt sich daraus eine ‘avarage-case performance’ von $O(n \cdot log(n))$ (vgl. H. W. Lang, *Sortierverfahren Heapsort*).

2.3.4 Insertionsort

Bei Insertionsort (engl. insertion: Einfügen, sort: sortieren) handelt es sich wiederum um einen relativ einfachen Sortieralgorithmus. Das Prozedere mit dem das Problem sortiert wird ist vergleichbar mit dem des Bubblesort. Es wird ein einzelnes Element betrachtet, ist es größer als sein nächster Nachbar so werden die Element getauscht. Das Element welches nun nach hinten verschoben wurde wird solange weiter nach hinten verschoben bis es nicht mehr kleiner ist als das nächsthintere Element. Dieser Vorgang wird für alle Elemente, beim ersten beginnend bis zum letzten, genau einmal durchgeführt. Daraus lässt sich erkennen, dass die Insertionsort-Methodik zum Einen eher für kleinere Problem geeignet ist, zum Anderen aber dennoch schneller ist als der Bubblesort-Algorithmus ist, da nach einem einzigen Durchlauf das Problem bereits gelöst ist. Insertionsort ist mit einer ‘avarage-case performance’ von $O(n^2)$ angegeben (vgl. H. W. Lang, *Sortierverfahren Insertionsort*).

2.3.5 Mergesort

Ähnlich wie der Quicksort-Algorithmus bedient sich auch der Mergesort-Algorithmus an dem übergeordnetem „Teile und herrsche“-Prinzip. Dabei fällt, anders als bei Quicksort, die meiste Arbeit auf das Zusammenführen der Unterprobleme aus. Dies spiegelt sich deutlich in dem Source-Code wieder. Ist das Problem vollständig in die kleinst mögliche Teilmenge, auch elementare Menge, zerlegt, so bricht die Rekursion ab und die Teillösungen werden wieder zusammen geführt (B. Voecking, *Taschenbuch der Algorithmen*, vgl. S.24). Gleichzeitig werden die Elemente der Teilmengen miteinander verglichen und an die entsprechende Stelle geschrieben (vgl. H. W. Lang, *Sortierverfahren Insertionsort*).

2.4 Entwurf der Oberfläche

Die Oberfläche, im folgenden GUI genannt, sollte zunächst so simpel wie möglich sein und den Betrachter auf das Wesentliche lenken: die Balkendiagramme in der Mitte der Applikation. Oberhalb sowie unterhalb der Balkendiagramme, jeweils abgetrennt durch einen feinen Strich, finden sich die wesentlichen Steuerelement wieder.

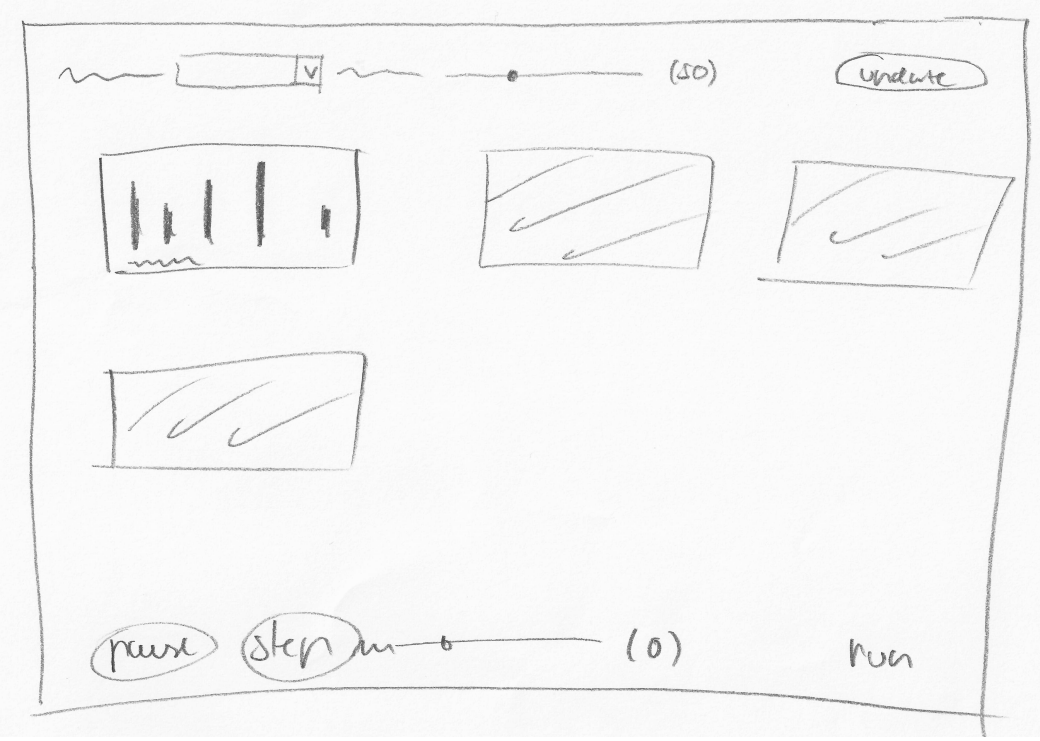

Abbildung 2.1: Entwurf auf Papier

Oberhalb der Balkendiagramme wird zunächst in einem Drop-Down-Menue die Art des Problems festgelegt. Die Auswahlmöglichkeiten umfassen:

- „Random“ - für eine zufällige Zahlenreihe
- „Reverse“ - für eine absteigend sortierte Zahlenreihe
- „Stairs“ - für eine stufenartige Zahlenreihe mit wenig einzigartigen Element
- „predefined“ - für eine immer gleichbleibende Zahlenreihe[1]

Rechts neben dem Drop-Down-Menue befindet sich ein Slider, mit dem man die größe des Problems definiert. Die einstellbaren Werte reichen von 10 bis 50 und lassen sich in Zehnerschritte verändern. Rechts daneben ist ein Button platziert, mit dem die Einstellungen auf die Balkendiagramme übertragen werden.

Darunter befinden sich die Balkendiagramme, die, je nach eingestellter Problemgröße, unterschiedlich breite Balken ausweisen. Unterhalb der Balkendiagramme ist die Legende platziert, welche die verschiedenen Operationstypen mit passenden Farbschattierungen gegenüberstellt. In dem unteren Teil des Fensters befinden sich weiter Steuerelemente. Auf der linken Seite gibt es ein Button für das Pausieren der laufenden Sortierungen. Wird pausiert, so kann der Button daneben benutzen werden um manuell jeden weiteren Operationsschritt einzeln durchzuführen. Rechts daneben befindet sich ein weierer Slider, mit dem intervallartig die Sortiergeschwindigkeit eingestellt werden kann. bschließend ist ein Button mit der Aufschrift „sort“ in der rechten unteren Ecke platziert, der nach getroffener „Voreinstellung“ betätigt werden kann, so dass alle Algorithmen mit der Sortierung beginnen.

Der Untererand der App beinhaltet desweitern eine Statusleiste die Aufschluss über den Zustand der Application selbst gibt. Abbildung 2.2 zeigt wie der Entwurf umgesetzt wurde.

2.5 Programmablauf

Startet mann die Applikation, so werden als erstes die Standardeinstellungen geladen: Problem „Random“, Problemsize 10, Interval 250 Millisekunden. Anschließende können die Einstellungen entsprechend dem eigenen ermessen verändert werden. Die Buttons „pause“ und „step“ sind inaktiv, sie lassen sich also noch nicht betätigen. Nur der Button „sort“ ist aktiv. Wird auf „sort“ geklickt, so kann beobachtet werden

[1]Mit der Auswahl predefinedïst sichergestellt, dass die Auswertungen vergleichbar bleiben.

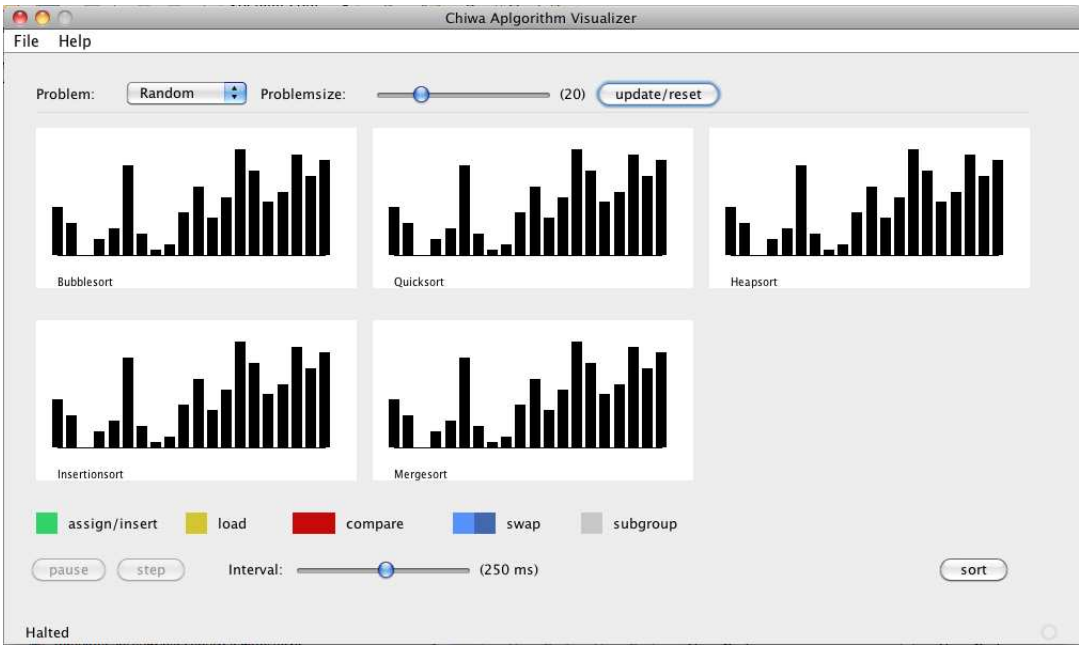

Abbildung 2.2: Java-Application nach dem Start

wie die Balken mit verschieden Farben hervorgehoben werden und sich das Balkendiagramm, je nach eingestellter Sortiergeschwindigkeit, schnell oder langsam sortiert. Nun ist der „pause“-Button aktiviert und der Sortiervorgang kann angehalten werden. Mit dem Button „step“ kann nun schrittweise weitersortiert werden. Zeitgleich steht im pauseButton „resume“. Angeklickt lässt dieser den Sortiervorgang wieder automatischen weiterlaufen. Ist dies geschehn, steht im resumeButton auch wieder „pause“. Zudem ist während des ganzen Sortiervorgangs der Button „sort“ deaktiviert.

Ist ein Balkendiagramm komplett sotiert und der Algorithmus beendet, so wird ein Linendiagramm über das Balkendiagramm gelegt, das die Position des Algorithmus im Vergleich zu den anderen beinhaltet, die Anzahl durchgeführter Operation sowie den Grad der Sortiertheit auf der Y-Achse und die Anzahl an Operationsschritten auf der X-Achse anzeigt.

An dem Punkt angekommen kann der Anwender nun entwerder noch einmal auf „sort“ drücken und die bereits sortierten Balken nocheinmal sortieren lassen. Oder er wiederholt den oben beschrieben Vorgang mit anderen Einstellungen. Falsche Eingaben kann der Nutzer dabei nicht treffen, alle Einstellungskombinationen sind denkbar und durchführbar. Einzig ein zu kurz gewähltes Interval kann auf langsamen Computern dazu führen, dass der Sortiervorgang nicht mehr flüßig aussieht.

Kapitel 3
IT-Konzept

3.1 Logische Einheiten

In den nächsten Abschnitten wird auf die verschieden Logischen Einheiten eingegangen. Logische Einheiten sind Komponenten oder auch Klassen, die eine klar diffenrzierbare Aufgabe innerhalb der Applikation haben.

3.1.1 App, main() und View

Die Klasse App ist bei der Erstellung eines neuen Projekts von Netbeans[1] erstellt worden. Die Klasse enthält die `main()`-Methode, welche zu dem Starten einer jeden Java-Applikation benötigt wird. Die Klasse App enthält eine Methode, welche die Klasse View instanziert.

```
1  /**
2  * At startup create and show the
3  * main frame of the application.
4  */
5  @Override protected void startup() {
6      show(new View(this));
7  }
```

Quellcode 1: Methode startup (App.java Z.18-20)

3.1.1.1 Der View

In dieser Klasse befinden sich alle Methoden, die zur Generierung der einzelen Steuerelemente dienen. Ferner werden alle Balkendiagramme und deren zugehörige Algorithmen instanziert. Zudem werden alle Eventhandler angelegt, mit denen die Anwendereingaben delegiert werden können.

[1] NetBeans IDE 7.0 (Build 201104080000)

3.1.2 Die RunnableSortingCollection

`RunnableSortingCollection`[2] Eine Instanz dieser Klasse beinhaltet alle Sortieralgorithmen, sofern programmatisch hinzugefügt, und delegiert eventuelle Steuerbefehle. Ein Beispiel ist das Starten des Sortiervorgangs. Die Klasse dient der Kommunikation zwischen Nutzeroberfläche, beziehungsweise Nutzer, und den Algorithmen.

3.1.3 Der SynchronizedSorter

Wie zu Anfang beschrieben braucht es Logik um zu gewährleisten, dass der Ablauf des Sortierens paralell und synchron über den ganzen „Existenzzeitraum“ der Applikation besteht. Erreicht wird dieses Verhalten durch die Nutzung von Native-Threads und der Tatsache dass seit Java 5 Threads ein Interkommunikationsmodel besitzen, die sich per Java programmieren lassen[3].

3.1.4 Der VisualFeedbackSorter

Eine der wichtigesten Komponenten der Applikation ist der `VisualFeedbackSorter`, welcher für die visuelle Repränstation der internen Sortiervogänge zuständigt ist. Wie auch der `SynchronizedSorter` bedient sich der `VisualFeedbackSorter` dem Decorator-Pattern (siehe 3.2.1) um seine Funktionalität der eigentlichen Algorithmusinstanz voranzustellen.

3.1.5 Die Klasse `AbstractSortingMechanics` und das Interface `Sorter`

Wie in Abb. 3.1 dargestellt sind das Interface `Sorter` und die Klasse `AbstractSortingMechanics` die Basis für den Decorator-Pattern. Zugleich bilden diese beiden Logik Einheiten die Basis für alle Klassen, die sich wie ein Algorithmus verhalten sollen (vgl. Abb. 3.3).

[2] „There are only two hard things in Computer Science: cache invalidation and naming things“ Phil Karlton

[3] siehe http://download.oracle.com/javase/tutorial/essential/concurrency/

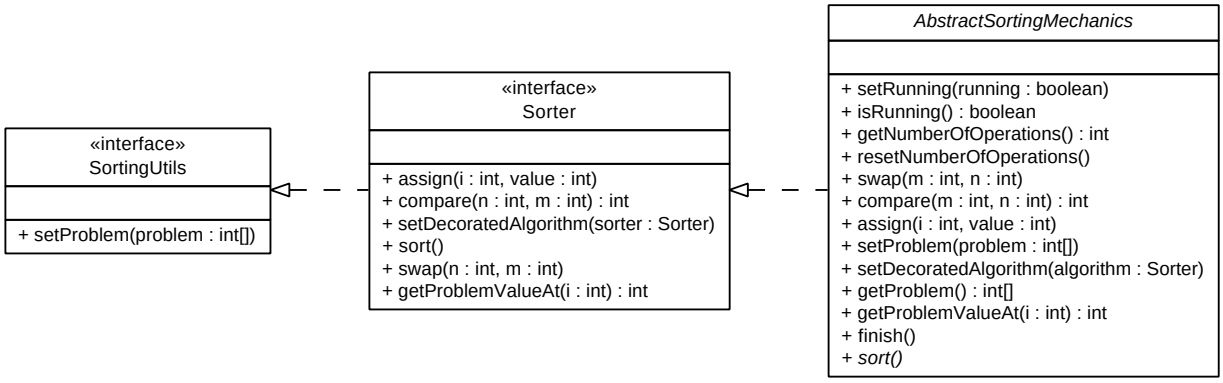

Abbildung 3.1: Klassenkarte von AbstractSortingMechanics.java und Sorter.java

3.2 Erläuterung wichtiger Entwurfsmuster

Entwurfsmuster benennen, abstrahieren und indentifiziern die Kernaspekte einer wiederkehrenden Herrangehensweise und schaffen somit wiederverwendbare objektorientierte Lösungen (vgl. Gamma u. a., *Design patterns: elements of reusable object-oriented software*, S.3).

3.2.1 Decorator-Pattern

> *“ Mit dem Decorator Pattern, in der deutschen Übersetzung naheliegenderweise Dekorierer genannt, lässt sich ein Objekt dynamisch um Fähigkeiten, auch Zuständigkeiten genannt, erweitern. Anstatt Unterklassen zu bilden und eine Klasse damit um Fähigkeiten bzw. Verhalten zu erweitern, lässt sich mit dem Einsatz des Decorator Patterns die Erzeugung von Unterklassen vermeiden ”(Sherzad,* Decorator Pattern in Java*)*

3.2.1.1 AbstractSortingDecorator

Diese Klasse bildet die Grundlage für den Decorator-Pattern und wird an `VisualFeedbackSorter` und `SynchronizedSorter` weitervererbt. Wird einen dieser Dekoratoren instanziert und das zu dekorierende Objekt an ihn übergeben, so verhält der Dekorator nach außen hin genau so wie das übergebene Objekt. Darin liegt die Stärke diese Entwurfsmusters welches sich in dem Konstruktor von `View` zeigt.

```
Sorter bubblesort    =
        new SynchronizedSorter(new VisualFeedbackSorter(new Bubblesort()));
Sorter heapsort      =
        new SynchronizedSorter(new VisualFeedbackSorter(new Heapsort()));
Sorter insertionsort =
        new SynchronizedSorter(new VisualFeedbackSorter(new Insertionsort()));
Sorter mergesort     =
        new SynchronizedSorter(new VisualFeedbackSorter(new Mergesort()));
```

Quellcode 2: Instanzierung und Dekorierung aller Algorithmen (View.java Z.69-73)

Nach wie vor verhalten sich alle Algorithmen nach außen hin wie ein `Sorter`, nur sind jetzt alle Methoden dekoriert. Wird nun von `bubblesort` eine Method aufrufen, so wird sie von `SynchronizedSorter` über `VisualFeedbackSorter` bis zu dem eigentlichen `Bubblesort` Algorithmus durchgereicht werden. Während dieser Prozedur kann jeder Dekorierer auch eigenen Code ausführen. Entscheident ist, dass `SynchronizedSorter` und auch `VisualFeedbackSorter` in den Konstruktoren den übergebenen Algorithmus in einem dafür vorgesehenen Feld speichen.

```
public abstract class AbstractSortingDecorator
  extends AbstractSortingMechanics {
    protected final Sorter algorithm;

    public AbstractSortingDecorator(Sorter algorithm) {
        this.algorithm = algorithm;
    }
}
```

Quellcode 3: Konstruktor (AbstractSortingDecorator.java Z.11-29)

Außerdem müssen die erbenden Klassen diese Feld zur „Redelegation“ der Methode benutzen, damit der Decorator-Pattern richtig implementiert ist (siehe Quellcode 4 Z.17).

```java
public class SynchronizedSorter extends AbstractSortingDecorator {

    public SynchronizedSorter(Sorter algorithm) {
        super(algorithm);
    }

    @Override
    public void swap(int m, int n) {
        synchronized (super.algorithm) {
            try {
                super.algorithm.wait();
            } catch (InterruptedException ex) {
                Logger.getLogger(SynchronizedSorter.class.getName())
            .log(Level.SEVERE, null, ex);
            }
        }
        algorithm.swap(m, n);
    }
```

Quellcode 4: Beispiel einer Dekoration (SynchronizedSorter.java Z.20-36.)

3.3 Erläuterung besondere Implementationen

3.3.1 Die Klasse `SynchronizedSorter`

Wie in Quellcode 4 und Abb. 3.2 bereits gezeigt erbt die Klasse von `AbstractSortingDecorator` und dieser wiederum von `AbstractSortingMechanics`. Die Methoden `swap`, `compare`, `assign`, `getProblemValueAt` werden mit einer speziellen Direktive implementiert(siehe Quellcode 4 Z.9-16). Das hat zur Folge, dass immer wenn eine dieser Methoden aufgerufen wird, der laufende Thread in einen wartendend Zustand versetzt wird (siehe Quellcode 4 Z.11). Dabei spielt das Objekt auf dem `wait()` ausgeführt wird keine Rolle. Der Thread wartet bis auf dem selben Objekt an einer anderen Stelle `notify()` ausgeführt wird. Durch diesen Aufbau ist der Ausführungsfluss zur Sortierung synchronisiert.

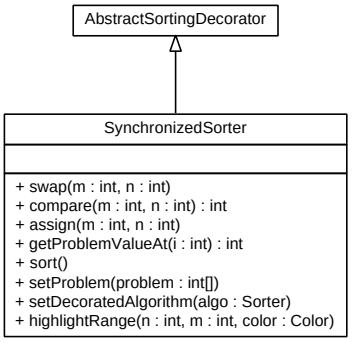

Abbildung 3.2: Klassenkarte

3.3.2 Die Klasse `VisualFeedbackSorter`

Diese Klasse baut zum Start der Applikation mit Hilfe der Charting-Bibliothek `jChart2D`, das Balkendiagramm für den übergebenen Algorithmus auf und speichert die Instanz in einem Feld. Gleichzeitig fügt es die Balken in den View ein. Wird eine dekorierte Methode aufgerufen, so wird das Chart angewießen die entsprechenden Balken, zum Beispiel bei einer dekorierten `compare` Methode, mit einer Farbe zu hinterlegen. Anschließend wird der Aufruf weitergereicht.

3.3.3 Die Klasse `Surveyor`

Die Klasse Surveyor kann die Sortiertheit eines Integer Arrays im Rahmen von 0.0 bis 1.0 berechnen. Dabei wird die Distanz eine Elements zu seiner eigentlichen Position bestimmt. Diese Entfernungen werden aufsummiert und durch die maximal möglich Gesamtdistanz geteilt. Bei einer länge von n ergibt dies eine maximale Entfernung von n^2. Die Methodik wird für die Auswertung am Ende eines jeden Sortiervorgangs benutzt.

3.4 Klassendiagramm

In dem gezeigten Klassendiagramm befinden sich alle Klassen und Interfaces, die in der Applikation verwendet werden. Die Klassen die keinen Bezug zur Sortierstruktur haben und für die Verdeutlichung der Zusammenhänge keine signifikante Rolle spielen sind nicht enthalten.

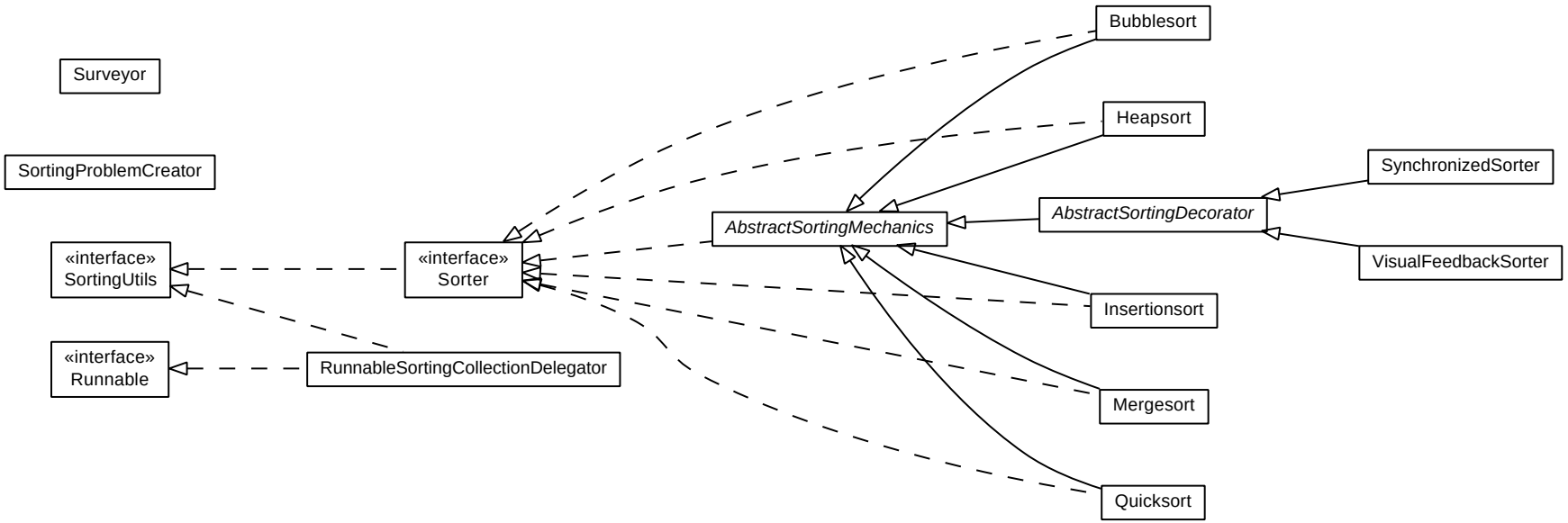

Abbildung 3.3: Klassendiagramm aller wichtigen Komponenten

Kapitel 4
Refelexion

Visualisierung

Bei der Betrachtung das Endergebnisses kann mann feststellen das dem Anwender leider nicht so einfach klar wird, wie die einzelnen Algorithmen tatsächlich funktionieren. Der Ansatz den ich gewählt habe eignet sich aus folgenden Gründen nicht dazu: Ich habe angenommen das es reichen würde die einzelen Operation, die ein Algorithmus im Regelfall durchführen kann zu isolieren und dann mit der Visualisierungslogik zu dekorieren. Daher ist der jeder Algorithmus ist ganz normal implmentiert, ohne spezielle Zusatzlogik. Die bedeutet im Umkehrschluss auch, dass mit der gegebenen Infrastruktur von Klassen und Interfaces fast jeder Algorithmus visualisiert werden kann.

Für sich genommen ist diese Herrangehensweise, denke ich, legitim und auch für simple Algorithme durchführbar. Jedoch das Ergebnis von der optischen Seite betrachtet nicht zufriedenstellend, da die Visualisierung der komplexeren Algorithmen eher wilkürlich wirken.

Nachträglich habe ich deswegen versucht weitere sinnvolle Teilvisualisierungen bei den komplexeren Algorithmen hinzuzufügen. Dennoch bleibt der Punkt, der eher geringen Anschaulichkeit bestehen, da man nur versteht wie der Algorithmus funktioniert wenn der Source-Code dafür gleichzeitg betrachtet wird. Und mit den optischen Herrvorhebungen verknüpft ist. Diese Verfahren habe ich verwendet um Passagen dieser Seminararbeit zu anschaulicher zu formulieren. Inbesondere bei dem Quicksort-Algorithmus hat mir die Visualisierung gut geholfen. Jedoch kann ich den Source lesen, was mann nicht von allen anderen behaupten kann.

Ein besser Ansatz

Den Fehler, wenn man überhaupt von einem Fehler sprechen kann, liegt klar in der Progammtechnischen orientierten Herrangehensweise. Davon ausgehen habe ich Strukturen programmiert, die es einfach machen jeden Algorithmus zu visualisieren. Die Vorüberlegung, einzelene Operationsschritte in einem größern Zusammenhang und im Zusammenwirkung mit anderen Operationsschritten sinnvoll visualisieren, hätte eventuell ein anschaulichers Ergebnis zur Folge.

Literatur

B. Voecking, H. Alt. *Taschenbuch der Algorithmen*. Springer, 2008.

Gamma, Erich u. a. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

H. W. Lang, None. *Sortierverfahren Bubblesort*. URL: http://www.inf.fh-flensburg.de/lang/algorithmen/sortieren/networks/bubble.htm.

— *Sortierverfahren Heapsort*. URL: http://www.inf.fh-flensburg.de/lang/algorithmen/sortieren/heap/heap.htm.

— *Sortierverfahren Insertionsort*. URL: http://www.inf.fh-flensburg.de/lang/algorithmen/sortieren/merge/merge.htm.

— *Sortierverfahren Insertionsort*. URL: http://www.inf.fh-flensburg.de/lang/algorithmen/sortieren/insert/insertion.htm.

— *Sortierverfahren Quicksort*. URL: http://www.inf.fh-flensburg.de/lang/algorithmen/sortieren/quick/quick.htm.

Sherzad, Rias A. *Decorator Pattern in Java*. URL: http://www.theserverside.de/decorator-pattern-in-java/.

Wikipedia. *Algorithmus*. URL: http://de.wikipedia.org/wiki/Algorithmus.

— *Quicksort*. URL: http://de.wikipedia.org/wiki/Quicksort.

Anhang A

Schlusserklärung

Ich erkläre hiermit, dass ich die Facharbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benützt habe.

München, den 3. Oktober 2011 Unterschrift (Kai Richard König)