Übungsblatt 4 (Block 1)

Prof. Dr. Olaf Hellwich und Mitarbeiter

Handsimulation und Implementierung von Sortierverfahren

Erinnerung: Bitte denken Sie an den abschließenden Block-Test.

Verfügbar ab:	**14.05.18**
Abgabe bis:	**21.-25.05.18**

Aufgabe 1: MySort

5 Punkte

Laden Sie zur Bearbeitung der Aufgaben das komprimierte Verzeichnis `Vorgaben.zip` von der Webseite der Veranstaltung herunter. Das Verzeichnis enthält die Dateien

filme1.dat	- Textdatei, die Daten von Filmen enthält (unsortiert).
filme2.dat	- Textdatei, die Daten von Filmen enthält (aufsteigend sortiert).
filme3.dat	- Textdatei, die Daten von Filmen enthält (absteigend sortiert).
Parser.java	- Java-Klasse zum Einlesen von Filmdaten.
Film.java	- Java-Klasse, die Filme repräsentiert.
TestMySort.java	- Java-Klasse zum Testen der Implementierung von Aufgabe 1.
TestQuickSort.java	- Java-Klasse zum Testen der Implementierung von Aufgabe 2.

Das Interface `Comparable` besitzt eine Methode `int compareTo(Comparable other)`. Der zurückgelieferte `int`-Wert `value` hat folgende Bedeutung:[1]

1. `value > 0` : dieses Objekt ist größer als das Objekt `other`
2. `value = 0` : dieses Objekt ist gleich dem Objekt `other`
3. `value < 0` : dieses Objekt ist kleiner als das Objekt `other`.

Die Klasse `Film` implementiert das Interface `Comparable`. Der Rückgabewert der Methode `int compareTo(Comparable other)` ist durch die lexikographische Ordnung der Filmtitel bestimmt.

- Ist das übergebene Objekt kein Film-Objekt (`instanceof`), so soll die Rückgabe 0 sein.
- Für die Klasse `String` existiert bereits eine `compareTo`-Implementierung, die anhand der lexikographischen Ordnung entscheidet.

Aufgabe:

1. Implementieren Sie den Rumpf der Methode `int compareTo(Comparable other)` der Klasse `Film`.
2. Erstellen, implementieren und kommentieren Sie eine Klasse `MySort`. Die Klasse soll die statische Methode

```
public static void mySort(Comparable[] f)
```

[1] In dem ersten Tutoriumsblatt haben wir das Prinzip des Interface *Comparable* schon einmal behandelt. Sollten hier also Verständnisprobleme auftreten, ist es ratsam, sich die Tutoriumsaufzeichnungen noch einmal anzuschauen.

besitzen, die das im Folgenden beschriebene Sortierverfahren realisiert.

- Das Array wird in einen bereits sortierten Teil (vorne, am Anfang leer) und einen noch nicht sortierten Teil (hinten, am Anfang das komplette Array) unterteilt.
- Durchlaufe nun den noch nicht sortierten Teil von vorne bis hinten und merke die Position des kleinsten Elements.
- Tausche das gefundene kleinste Element mit dem ersten Element des unsortierten Bereichs.
- Der sortierte Bereich ist nun ein Element größer als vorher, der unsortierte Bereich ein Element kleiner.
- Wiederhole, bis der noch nicht sortierte Teil leer ist.
- Das Sortierverfahren soll zusätzlich die Anzahl der ausgeführten Aufrufe der Methode `compareTo` und die Anzahl der vorgenommenen Vertauschungen auf der Konsole ausgeben.

3. Beantworten Sie die folgenden Fragen:
   a) Unter welchem Namen ist dieser Sortieralgorithmus gemeinhin bekannt?
   b) Ist dieser Sortieralgorithmus stabil? Warum/warum nicht?
4. Testen Sie Ihr Programm mit Hilfe der Klasse `TestMySort.java`. Gehen Sie dabei wie folgt vor:
   a) Sortieren Sie die Filme (jeweils für jede der drei Dateien filme1.dat filme2.dat und filme3.dat) anhand ihrer Titel aufsteigend.
   b) Geben Sie nach jeder Sortierung alle Filme mit Hilfe der Methode `toString()` der Klasse `Film` auf der Konsole aus.
   c) Beantworten Sie folgende Frage: Bei welchem vorliegenden Array (unsortiert, aufsteigend sortiert, absteigend sortiert) ist MySort am schnellsten und bei welchem am langsamsten? Begründen Sie Ihre Antwort.

Aufgabe 2: Quicksort — 5 Punkte

Für die Bearbeitung dieser Aufgabe verwenden ebenfalls Sie die in Aufgabe 1 genannten Vorgabedateien.

Erstellen Sie die Klasse `SortierenQuicksort`. Sollten Sie Aufgabe 1 noch nicht bearbeitet haben, implementieren Sie zuerst den Rumpf der Methode `int compareTo(Comparable other)` der Klasse `Film`, siehe Aufgabe 1.

Die Klasse `SortierenQuicksort` soll die statische Methode

```
public static void quicksort(Comparable[] f)
```

besitzen, die das Sortierverfahren Quicksort realisiert. Das Sortierverfahren soll zusätzlich die Anzahl der ausgeführten Aufrufe der Methode `compareTo` und die Anzahl der vorgenommenen Vertauschungen auf der Konsole ausgeben. Implementieren Sie den Algorithmus in der **In-Situ**-Variante (siehe Tutorium 4).

Hinweis: Das Zählen der Methodenaufrufe ist bei der Implementierung des Quicksort-Algorithmus vielleicht einfacher, wenn Sie dafür statische Klassenattribute verwenden.

Die Ermittlung des Indexes des Pivotelements im [Teil-]Array soll dabei zufällig erfolgen. Zur Erinnerung: Eine natürliche Zufallszahl zwischen 0 und n kann unter Verwendung der Zufallsfunktion der Standardbibliotheken wie folgt erzeugt werden: `int zufallszahl = (int)(Math.random() * n);`

Testen Sie Ihr Programm mit Hilfe der Klasse `TestQuicksort.java`. Gehen Sie dabei wie folgt vor:

1. Sortieren Sie die Filme (jeweils für jede der drei Dateien filme1.dat, filme2.dat und filme3.dat) nach aufsteigender Reihenfolge ihrer Titel (lexikographische Sortierung).
2. Geben Sie nach jeder Sortierung die Filme mit Hilfe der Methode `toString()` der Klasse `Film` auf der Konsole aus.

3. Beantworten Sie folgende Frage: Bei welchem vorliegenden Array (unsortiert, aufsteigend sortiert, absteigend sortiert) ist der Quicksort-Algorithmus am schnellsten bzw. am langsamsten? Nennen Sie die Gründe dafür.